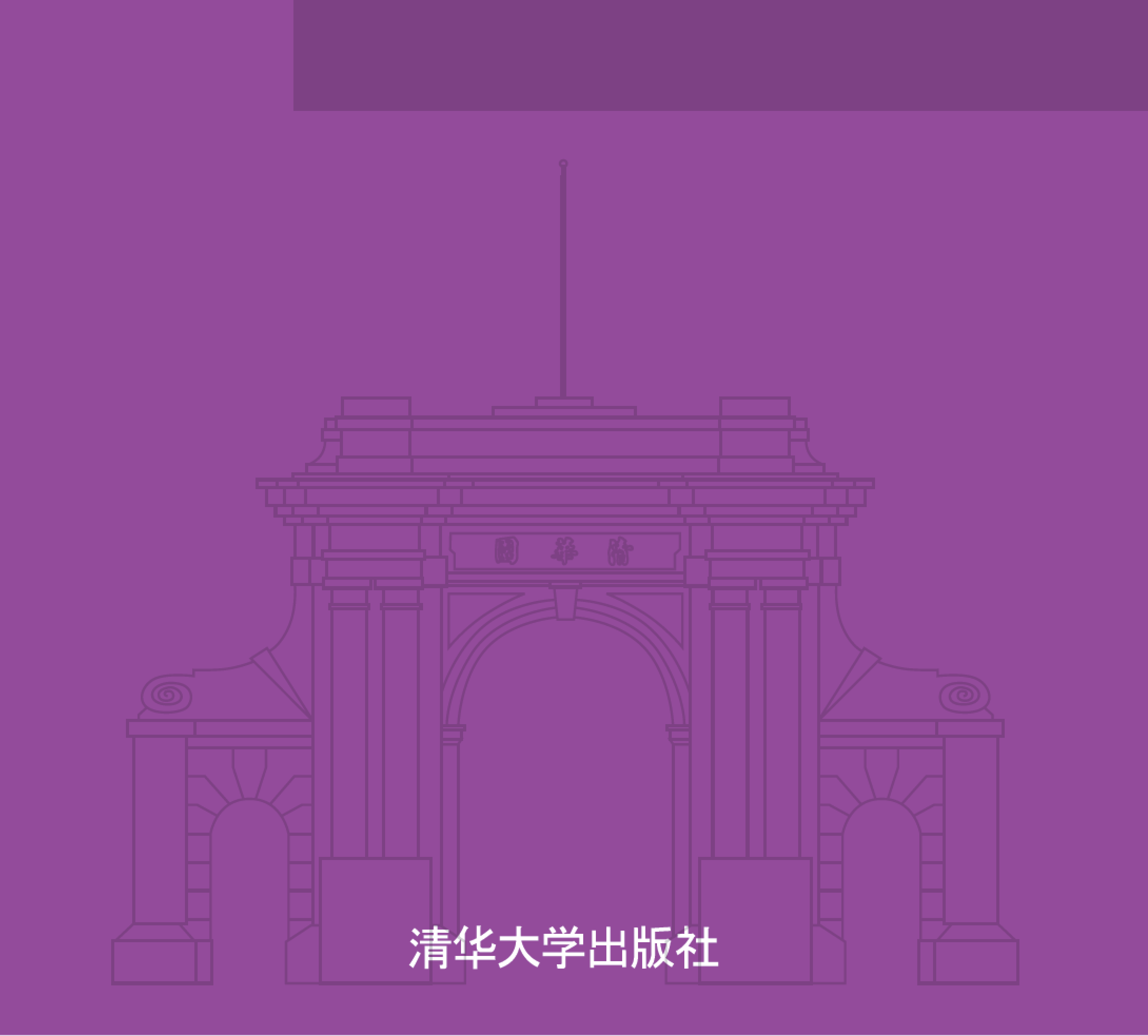
清華大學 计算机系列教材

陈震 编著

互联网安全原理与实践



清华大学出版社

清华大学计算机系列教材

互联网安全原理与实践

陈　震　编著

清华大学出版社

北　京

内 容 简 介

本书主要阐述网络安全的原理与技术。全书共分为三大部分：第一部分着重讲述网络安全的原理及本质，从网络安全的定义、网络安全的产生以及网络的基本原理和网络的攻击几个方面进行了详尽的阐述；第二部分结合 Untangle-UTM 系统，从 UTM 的配置入手，讲解了 Untangle-UTM 的主要功能模块，从中很好地介绍了相关的网络安全的基本知识；第三部分是具体的开发，其中有整体系统的分解与代码介绍，如反垃圾邮件模块、反钓鱼攻击模块、反病毒模块的设计以及流量分析与图形展示，便于读者从事开发和研究工作。

本书经过 7 年实际教学使用，选取的内容力求丰富全面，基本概念的讲解细致，深入浅出，简单明白；各种操作的讲解，都配有大量的实例操作和详尽的解析；非常适合于初学者，同时结构清晰，兼顾实用性与理论性，也有利于高年级本科生和研究生有选择地学习。

本书可作为计算机专业及相关专业的教材，也可作为计算机网络安全爱好者的参考用书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

互联网安全原理与实践/陈震编著．—北京：清华大学出版社，2014
清华大学计算机系列教材
ISBN 978-7-302-34008-9

Ⅰ．①互…　Ⅱ．①陈…　Ⅲ．①互联网络－安全技术－高等学校－教材　Ⅳ．①TP393.408

中国版本图书馆 CIP 数据核字(2013)第 234328 号

责任编辑：白立军
封面设计：常雪影
责任校对：白　蕾
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：北京密云胶印厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：16.25　　**字　　数**：387 千字
版　　次：2014 年 1 月第 1 版　　**印　　次**：2014 年 1 月第 1 次印刷
印　　数：1～2000
定　　价：31.00 元

产品编号：048842-01

序

“清华大学计算机系列教材”已经出版发行了30余种，包括计算机科学与技术专业的基础数学、专业技术基础和专业等课程的教材，覆盖了计算机科学与技术专业本科生和研究生的主要教学内容。这是一批至今发行数量很大并赢得广大读者赞誉的书籍，是近年来出版的大学计算机专业教材中影响比较大的一批精品。

本系列教材的作者都是我熟悉的教授与同事，他们长期在第一线担任相关课程的教学工作，是一批很受本科生和研究生欢迎的任课教师。编写高质量的计算机专业本科生(和研究生)教材，不仅需要作者具备丰富的教学经验和科研实践，还需要对相关领域科技发展前沿的准确把握和了解。正因为本系列教材的作者们具备了这些条件，才有了这批高质量优秀教材的产生。可以说，教材是他们长期辛勤工作的结晶。本系列教材出版发行以来，从其发行的数量、读者的反映、已经获得的国家级与省部级的奖励，以及在各个高等院校教学中所发挥的作用上，都可以看出它的社会影响与效益。

计算机学科发展异常迅速，内容更新很快。作为教材，一方面要反映本领域基础性、普遍性的知识，保持内容的相对稳定性；另一方面，又需要紧跟科技的发展，及时地调整和更新内容。本系列教材都能按照自身的需要及时地做到这一点。如王爱英教授等编著的《计算机组成与结构》、戴梅萼教授等编著的《微型计算机技术及应用》都已经出版了第四版，严蔚敏教授的《数据结构》也出版了三版，使教材既保持了稳定性，又达到了先进性的要求。

本系列教材内容丰富，体系结构严谨，概念清晰，易学易懂，符合学生的认知规律，适合教学与自学，深受广大读者的欢迎。系列教材中多数配有丰富的习题集、习题解答、上机及实验指导和电子教案，便于学生理论联系实际地学习相关课程。

随着我国进一步的开放，我们需要扩大国际交流，加强学习国外的先进经验。在大学教材建设上，我们也应该注意学习和引进国外的先进教材。但是，“清华大学计算机系列教材”的出版发行实践以及它所取得的效果告诉我们，在当前形势下，编写符合国情的具有自主版权的高质量教材仍具有重大意义和价值。它与国外原版教材不仅不矛盾，而且是相辅相成的。本系列教材的出版还表明，针对某一学科培养的要求，在教育部等上级部门的指导下，有计划地组织任课教师编写系列教材，还能促进对该学科科学、合理的教学体系和内容的研究。

我希望今后有更多、更好的我国优秀教材出版。

清华大学计算机系教授，中国科学院院士

张钹

前　　言

互联网在信息社会生活中扮演着越来越重要的角色，互联网的使用领域也拓展到了各个行业，如工业控制网、物联网、车联网，以及星际间联网。这个边界，随着人类的认识还在不断地延伸。

赛伯空间(Cyberspace)是由互联网连接的信息系统组成的信息空间，与目前现实四维空间对应。互联网安全是由互联网连接的信息系统组成的赛伯空间的控制权的博弈。

当前，人类对互联网连接的信息系统的控制权的争夺非常激烈，世界各国都在争夺基于互联网的信息空间的控制权。例如，美国的“爱因斯坦计划”被用以发现美国所有政府网站上的入侵、监控网络系统安全，很可能是一种大规模应用人工智能技术的网络监控技术。又比如“棱镜计划”，也是通过控制互联网上的信息系统，获取情报信息的一种国家战略。

同时，随着平板电脑、智能手机等移动终端的销售量和使用量日益剧增，移动应用软件的安全问题也日益严重，给移动互联安全带来了新的挑战。例如，安卓应用软件的安全问题越来越严重；任何第三方都可以随意开发应用软件并上传到应用平台，但各个应用市场的安全管理标准存在巨大差异，而从应用市场上下载的应用软件可以轻易地被篡改、添加恶意内容后重新上传到应用平台。与此同时，基于 Windows 的病毒程序可以轻松地移植到安卓的应用程序中。这些都使得移动终端用户的安全受到极大威胁。

本书正是在这种环境和背景下写作而成的。本书主要阐述网络安全原理与技术，通过实际系统操作与系统剖析为手段，介绍了相关网络安全的基本知识。全书通过三个部分，依次循序递进，介绍计算机网络安全原理，UTM 系统操作实践，UTM 系统功能剖析和代码介绍，如反垃圾邮件模块、反钓鱼攻击模块、反病毒模块的设计、流量分析与图形展示。

本书内容经过 7 年实际教学使用，选取的内容力求丰富全面，基本概念的讲解细致，深入浅出，简洁明了。各种操作的讲解，都配有大量的实例操作和详尽的解析。本书编者长时间在网络安全领域从事研究，精通密码学、可信计算、互联网安全和软件可信性分析，对本书投入了巨大精力。本书语言精练，内容丰富，实用性强，可以让读者在较短时间内全面把握互联网安全，并应用于实际工作中。

本书承蒙国家自然科学基金项目(下一代互联网安全与隐私关键性技术的研究)、国家 863 目标导向项目(一体化网络数据深度安全检测与分析技术与系统)和国家 973 项目(未来互联网体系结构与机制研究)支持。

本书还获得了英特尔公司的“基于 IXP 网络处理器的并行布隆过滤器(BloomFilter)实现的反蠕虫病毒系统”、“基于 IXP 2350 网络处理器的可信网络连接系统的设计和实现”、“基于云计算的系统漏洞分析与攻防平台”等项目支持。

感谢清华大学计算机系的姜欣博士、罗安安博士、彭冬生博士、张闰华、李国栋、陈志博、黄石海、韩阜业、许宏峰、石希、闵二龙、李军等参与系统研究与开发工作，谢谢他们的付出！

感谢清华大学信息技术国家实验室李军教授、林闯教授、薛一波教授、汪东升教授等的支持和鼓励。

编　者

2013 年 9 月

目　　录

上篇　互联网安全原理

中篇 互联网安全平台配置

下篇　网络安全平台开发

上　　篇

互联网安全原理

第1章　网络安全导论

1.1　网络安全

世界不安全，网络会安全吗？“网络社会”(Cyber Society，也叫赛伯空间)是在由计算机网络提供的低成本的信息通信、存储和传播功能的信息基础框架上(技术可能性)，由人类社会中的网民虚拟出来的一个社会空间，这个空间活跃的社会角色都落实在计算机的代理角色上。

网络社会这个虚拟的社会折射了很多现实社会的影子，反映了很多现实社会中不能获得诉求。因此，社会的行为，如互助、竞争、攻击等；社会的关系，如伙伴、朋友、圈子等；社会的愿望，如正义、公平和稳定等，这些社会特征一样会反映到这个虚拟空间上，最后落实到提供信息基础框架的计算机上的软件和网络设备上。

现实社会中有如下典型的安全事件。

(1) 2001年的9·11事件，恐怖分子劫持了2架客机撞击美国世贸中心双子大楼，导致世贸中心夷为平地，美国社会陷入了对国家安全的担忧之中。

(2) 2008年次贷危机，雷曼兄弟倒闭，美林破产，花旗大幅贬值几近崩溃，导致全球陷入经济危机，数百万人失业，经济衰退。

(3) 2009年金融危机进一步恶化，中国和国际社会一道对美元作为储备货币所引发的经济安全问题表示担忧。

(4) 2013年6月，美国“棱镜计划”曝光，引发了各国对网络空间军事竞赛的担忧。

……

世界动荡不安，可以想象网络上会风平浪静吗？

1.2　互联网安全风险无处不在

人们平时打开计算机，主要是登录学校信息门户网站，收收邮件，偶尔用网银支付一下网费，或者下载一些软件，下载电影看看等，感觉不到这里有什么安全风险。

殊不知，来自网络的安全威胁从计算机连接网络的这一刻开始就已经如影随形了，例如，机器获取的IP地址可能是一个私有DHCP服务器放出来的，下载的软件中可能被植入病毒，网站中有可能嵌入木马，接收的邮件中可能有“钓鱼”邮件等，互联网安全的风险无处不在。

1.3 人类社会的安全理念

从身边的社会中可以看到互联网安全的影子。

1. 安全的定义

安全(Security)的字面意思是免于风险和伤害。

本质上“安全”是一组物质设施,一种社会精神意识和个体的心理感觉。人的安全感来自什么?衣食无忧,家庭幸福,周围环境受我控制,职业和社会状况可确定可控(如公务员)。社会安全感来自什么?犯罪率低,社会稳定,公民都有好的社会保障体系和福利体系。国家安全感来自什么?民富国强,拥有最先进的武器。

2. 提高安全的手段

为了保障安全,在人类社会中应当做到如下几点。

(1) 了解威胁和风险。对威胁进行防范,要“未雨绸缪”时时监控;对攻击进行阻挡,要“防御进攻”。

(2) 提高自身的对抗能力。打疫苗,提高免疫能力,对危机的反应能力,对危机的控制能力。

(3) 隔离是保护安全的重要手段(防御)。如防盗门锁、禁闭室、监狱、黑名单、经济封锁等。

(4) 增强对威胁的控制性(防御走向进攻)。知己知彼,分析自身的脆弱性和黑客攻击方法及手段,主动防御。

(5) 惩罚手段。经济处罚,限制人身自由等惩罚手段。

3. “适度”安全

绝对安全状况是不存在的,因此追求安全是无止境的。安全的需求,基于人的现实需求;满足现实需求,基于人的心理需求;最终要满足人的心理需求。

对于安全,可以引入风险控制模型,如图1-1所示。举个例子,某些地区晚上出门是很危险的,要保障安全,晚上就尽量不要出门,以避免被人打劫;实在要出门,带个防身工具,如棒球棍,也可以减缓被人打劫的风险;最好几个人一起出门,可以在被人打劫的时候转移风

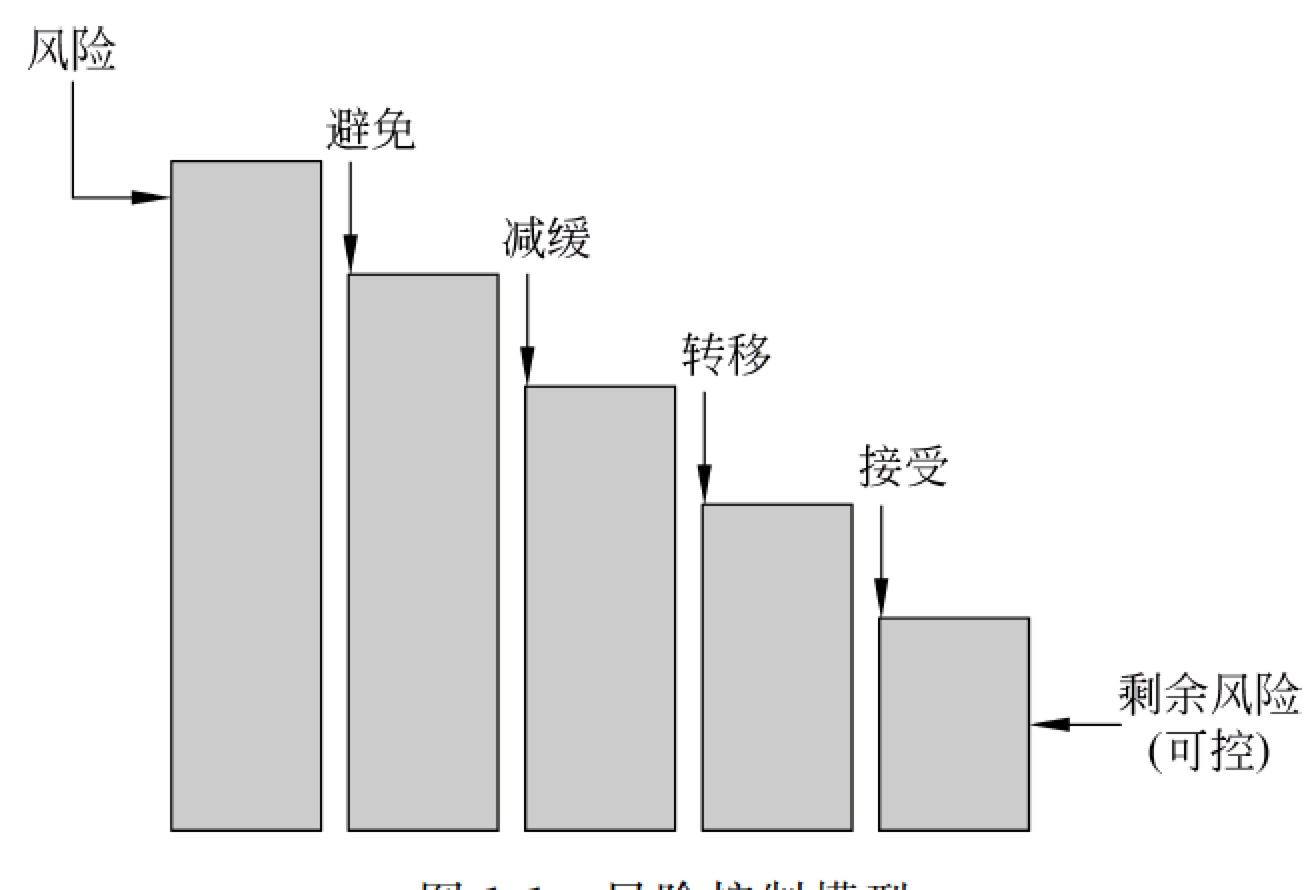


图1-1 风险控制模型

险;最后依然有风险,觉得可以承担,就出门吧。

安全产品对应的风险并不一定发生,这和买商业保险是一样的。因此,制造不安全气氛,到处发布各种安全事件损失以兜售安全产品的现象也会经常发生,这就是"安全"讹诈问题。安全资源的过度配置可能导致"过度"安全,为了保护1千元的资产,却需要花1万元的安全产品投资,是不合理的。对风险的评估是配置安全资源的依据,也要避免因小失大的问题。但是对涉及国家安全的机密数据和文档,其价值是不能用经济利益来衡量的。另外,人是整个安全保障体系中最重要的环节,因此即使进行了安全投资,也要在管理等其他人为因素上多下功夫,确保排除人为因素带来的安全隐患。

第 2 章　网络安全技术知识

2.1　互联网安全学科全景

网络安全是信息安全、计算机系统和计算机网络学科相关联的综合学科，因此涉及面广，工程性强，如图 2-1 所示。

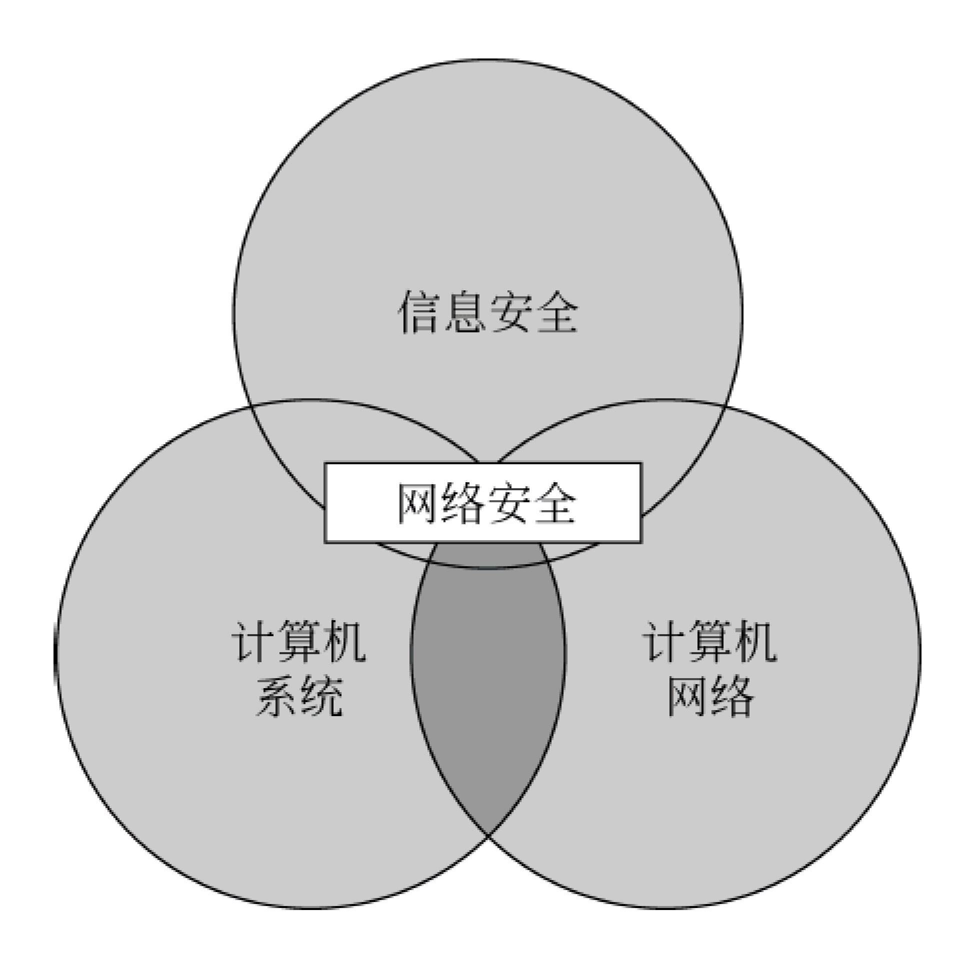


图 2-1　网络安全涉及计算机系统、计算机网络和信息安全

要了解互联网安全的原理，首先要了解计算机系统。

2.2　计算机系统

计算机系统由硬件（Hardware）系统和软件（Software）系统组成，根据不同的应用场合，计算机系统的硬件和软件配置将会有很大的差异，计算机系统形态如下。

（1）嵌入式计算机：手机、PDA、工控设备、家电控制器。

（2）个人计算机：个人电脑、上网本。

（3）中型计算机：服务器（文件、图像处理、金融交易等）。

（4）超级计算机：天河机等。

与计算本质相关的理论是计算理论（数学），主要用来研究计算机能做什么（计算的可行性），需要多大的时间和空间开销（计算复杂性），如何解决某些特定问题的过程与步骤（算法）和如何为速度、空间或者能耗等进行优化。

计算机系统首先是人造系统，是一个工程学适用的对象。从工程学的角度看，计算机系统架构和其他建筑工程没有本质的区别，都是采用基本组件按照系统结构进行搭建，都将尽量地复用已有的功能组件，减少成本。在系统架构的过程中，为了克服创建大系统的复杂性

问题，都会进行等级化、组件化，便于使用工具、分工协作等。因此计算机系统带有演进性、缺陷性等人类的社会特征。

计算机系统的发展也体现了人类在技术上不断追求更高、更快、更强的奥林匹克精神。当前的计算机系统也在不断演进。

2.2.1 计算机硬件

计算机硬件是一组集成电子器件和一些电子器件通过电子线路板集成的，计算机硬件具有明显的组织架构(如运算器/存储/显示/连接)，处理器有 Intel Core Duo 和 Core 2 系列、AMD Barcelona 和 Istanbul 系列、IBM Power PC 等，存储器有高速缓存 Cache、Flash、内存(Memory)、磁盘存储器，外部设备有输入/输出总线、鼠标、键盘、显示接口、音视频接口等。

与计算机硬件相关的理论科学有微电子物理、电磁学、材料学等。

计算机硬件体系结构设计是一种艺术，与人类社会结构相一致，其特征也包括人造系统(等级化、组件化、经济性、演进性、缺陷性等)。

个人电脑主板架构(CPU 前端总线＋北桥＋南桥)如图 2-2 所示。

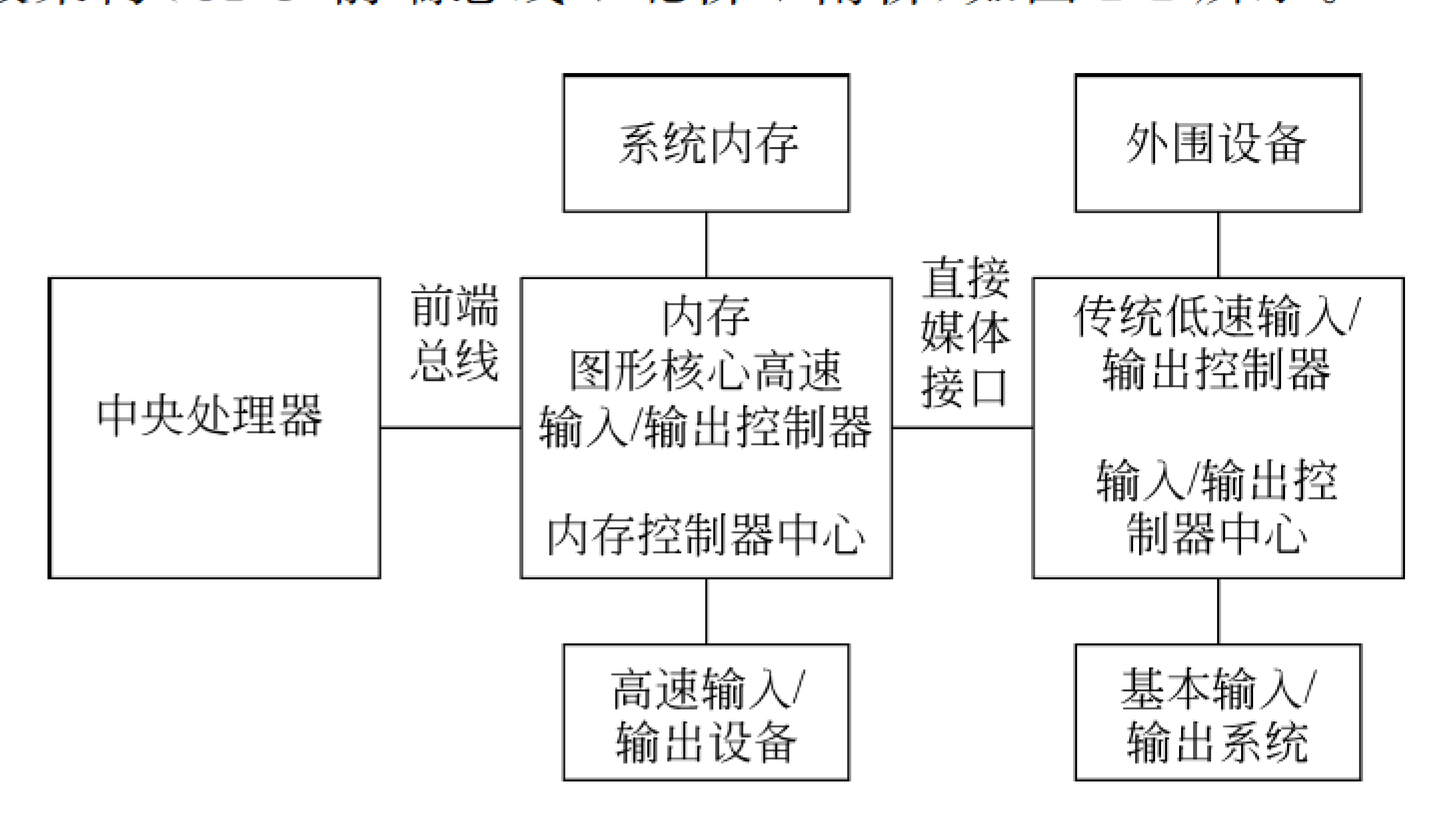


图 2-2　个人电脑主板架构(Intel 公司)

个人电脑主板高速互连如图 2-3 所示。

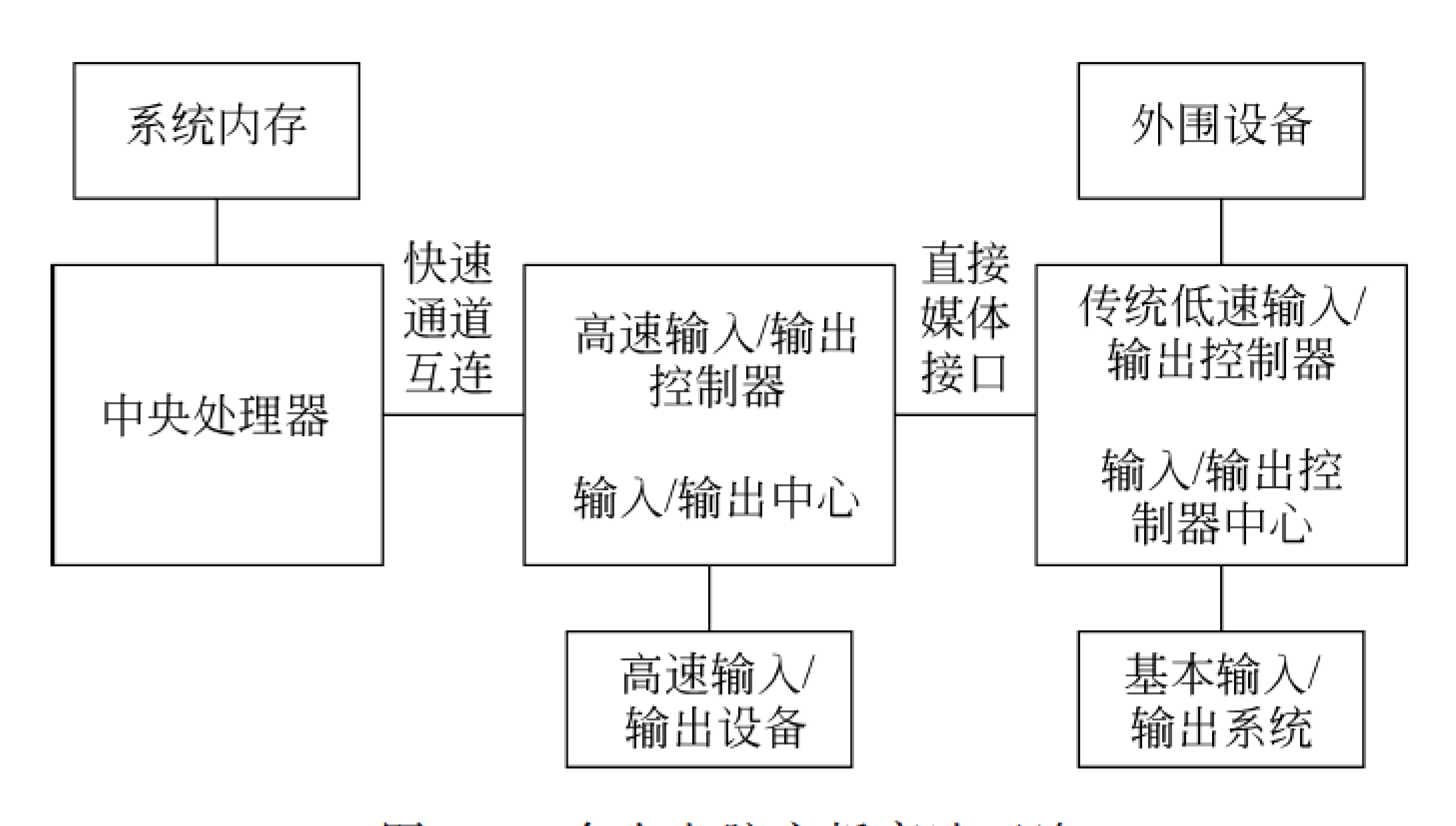


图 2-3　个人电脑主板高速互连

32 核处理器如图 2-4 所示。



图 2-4　32 核(128 线程)处理器

2.2.2　计算机软件

怎么生成计算机软件呢？一般来说，计算机软件是采用计算机工具软件制作出来的。对于软件要完成的功能，先用计算机程序语言(如汇编语言、C、C++、Java、C＃、Erlang、Perl、Python、JavaScript 等)编写出来，然后调用计算机程序类库 API，采用编译器(如 GNU、VC、Intel、SGI 等)生成可执行的代码。

软件设计需要软件架构，尤其是操作系统(Operating System)和应用软件(Application Software)，划分组件和功能、进程和线程。

目前基于 Web 的软件越来越流行，富客户端 RIA 的应用使得传统 C/S 结构出现了对等的趋势，相当部分的计算任务会转移到本地机器来实现。比较引人注目的是 Google 的 Native Client 技术使得在浏览器 Chrome 中运行二进制代码几乎和在本机上运行的软件一样快，这样就大大削弱了传统桌面软件霸主 Microsoft 在软件方面的优势地位。

软件体系结构设计是一种艺术，目前不同的操作系统，如 Berkeley FreeBSD、SUN Solaris、IBM AIX、Linux 等，其系统架构也是大有差异的。因此计算机软件的特征也包括人造系统的社会特性，如等级化、组件化、演进性、缺陷性等。常见的系统架构如图 2-5～图 2-7 所示。

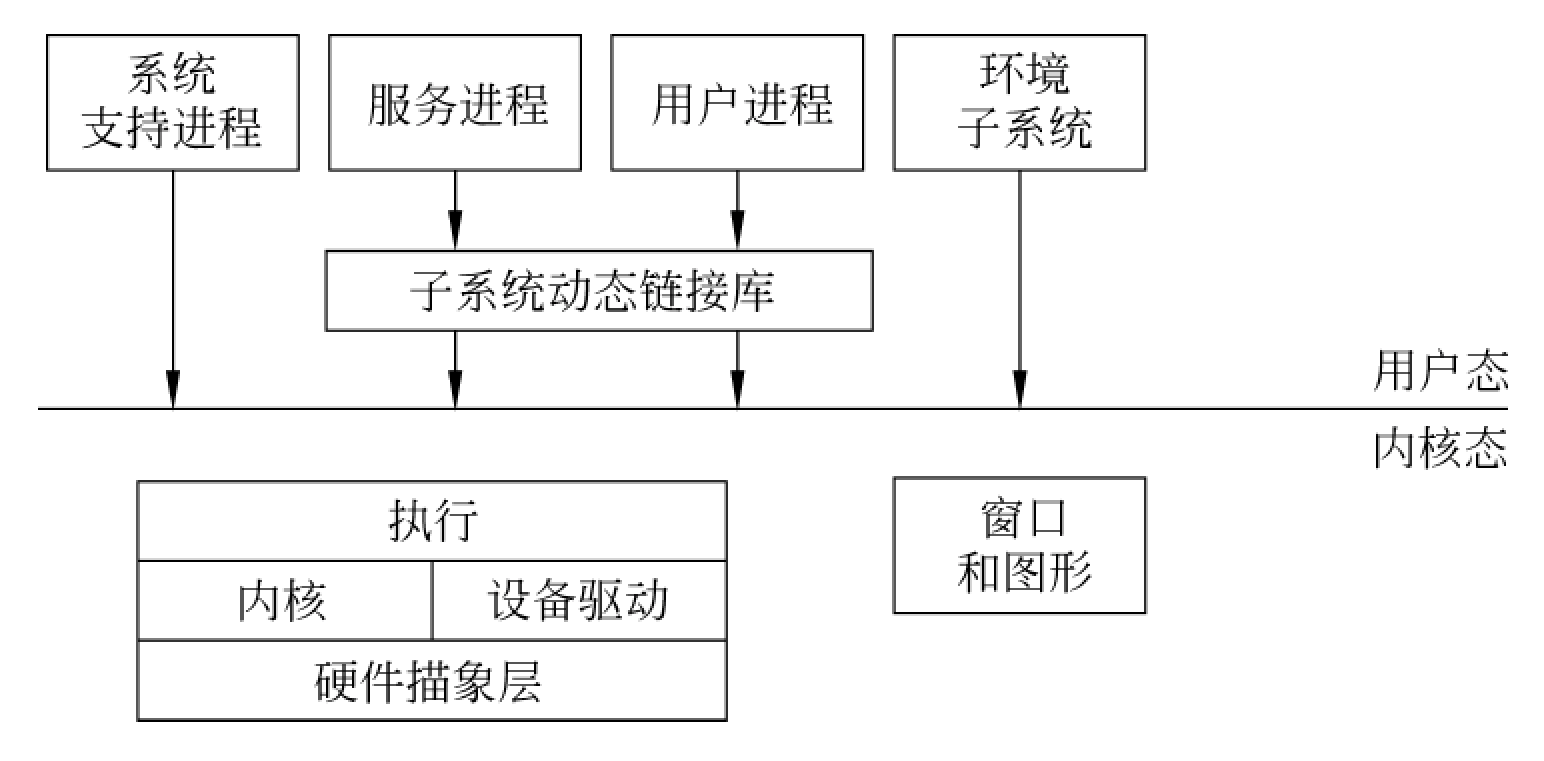


图 2-5　Windows XP 系统结构

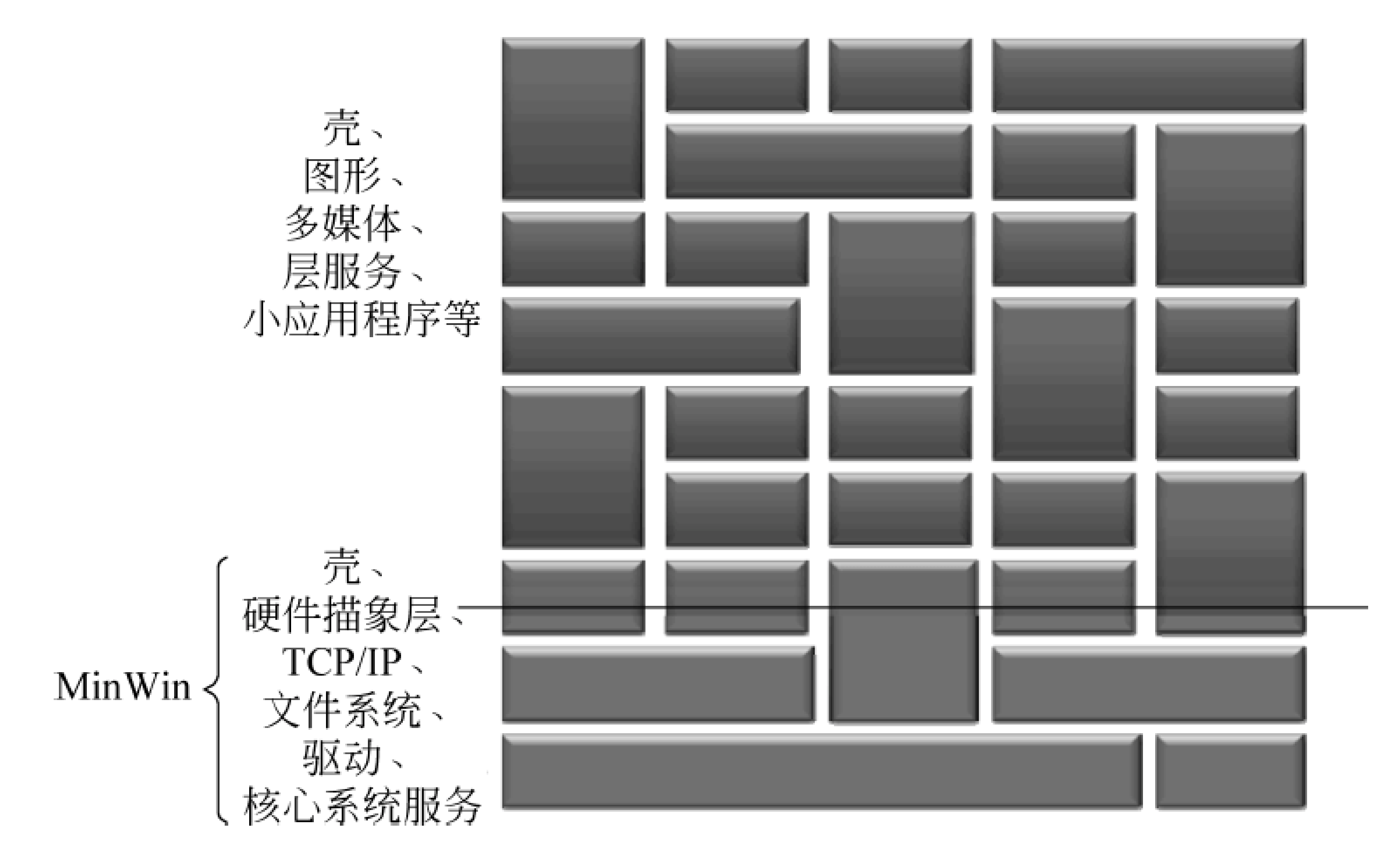


图 2-6　Windows 7 & Server 2008 MinWin Kernel

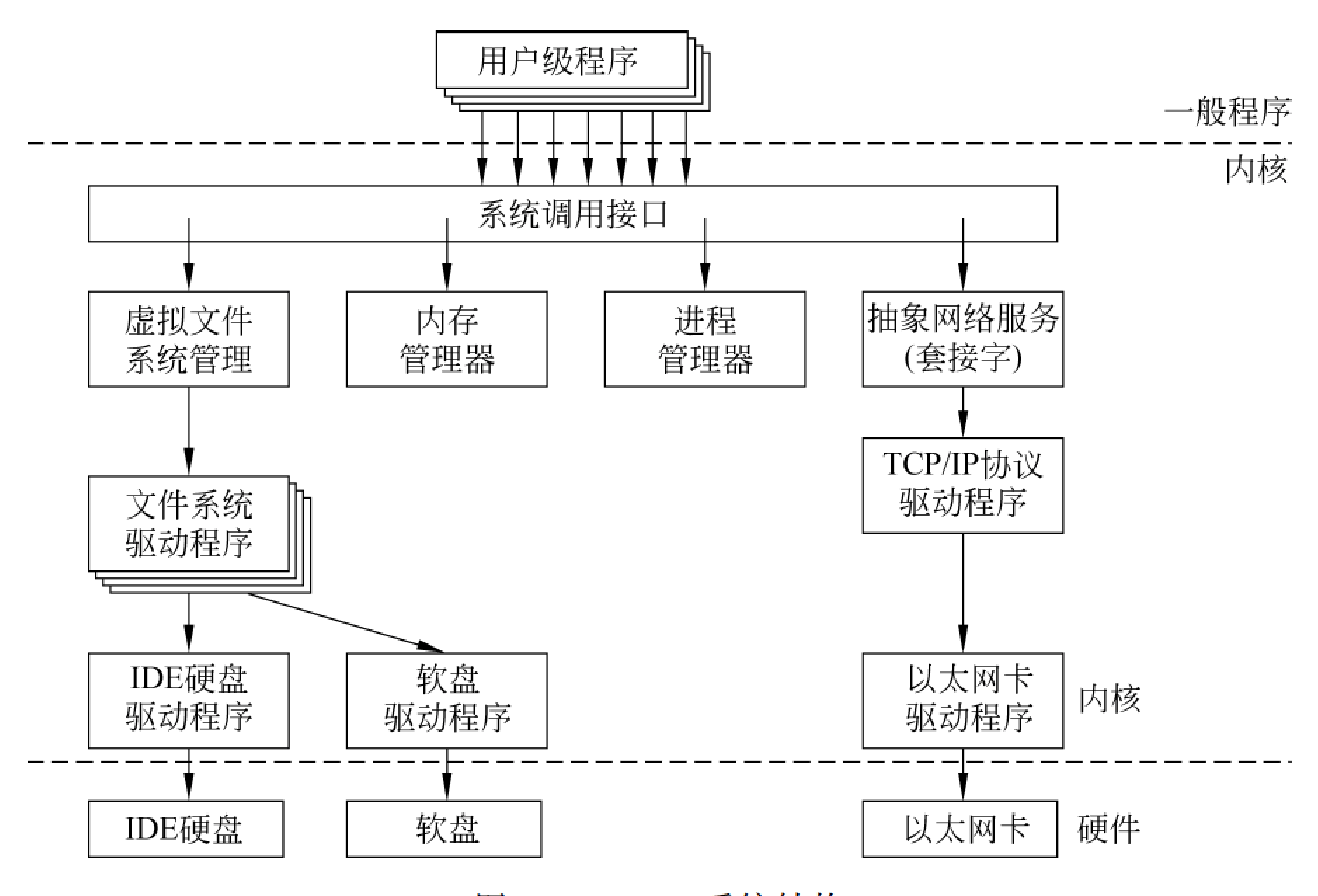


图 2-7　Linux 系统结构

2.3 计算机系统产业

计算机系统产业是一个生态系统(Ecosystem),其中的大公司如下。

芯片设计商:Intel、AMD、NVIDIA、VIA(威盛电子)、TI(德州仪器)、NS(美国国家半导体)等。

芯片制造商:台积电、Intel、MTK(联发科)、中芯国际等。

主板集成商:Intel、华硕、Acer、富士康等。

软件提供商:Microsoft、IBM、Oracle 等。

计算机系统产业是一个完整的工业链(Industrial Product Chain),有上中下游产品之分,下游厂商集成上游厂商生产的组件构造新的组件,最后组成服务产品,向终端零售商供货,最后交付给实际用户。

2.4 计算机网络

作为人类语音通信的电话网络早就存在(自 Alexandra Graham Bell 发明电话始,有 100 多年的历史),其他如电报网络也有很长的历史。从 1946 年的电子计算机出现,将计算机系统连接起来的共享计算/存储等资源的需求,是计算机网络产生的直接原因。

计算机网络是通过网络设备将独立异构计算机系统连接起来,完成计算机系统之间的资源共享。要解决的一个核心问题是需要定义一套原语(Primitive),即通信协议(如当前互联网采用的 TCP/IP),通信协议让不同的异构计算机之间能够交换数据。此外,计算机网络要研究如何有效地连接计算机系统,因此需要通信技术作为基础光通信数据通信。通信技术的革命也为计算机网络的普及打下了基础。

2.5 通 信 网 络

传统的通信网络负责信息传输和交换,通信网络运营商主要是作为信息的通道商,收取交通费用,并不提供内容,如电信网络、X.25 数据网络等。随着通信技术的突飞猛进,信息通信的通道价值日益降低,而通信内容的价值比较高。互联网最吸引人的是 Web 内容和各种音视频等,提供了丰富的内容,逐渐取代了一些传统的媒体形态,因此互联网已不再单纯是作为网络存在,而是作为一种新的内容方式而存在,人们平时说的“上网”,其实就是后者的形式。

3G 网络的兴起,为未来将数据通信进一步发展到普适的地步打下基础,以中国移动为代表的传统通信网络运营商为互联网接入提供丰富带宽以后,这些运营商将沦为“通道”的角色,很多服务提供商将会倒闭。因此这些运营商目前也大力开发内容,以提高网络的“粘性”。

2.6 计算机网络产业

计算机网络产业也是一个生态系统,其特征是网络系统是一个基础设施,基于计算机系统,每个组件都是计算机系统。

由组件构造新的组件，最后组成服务产品，符合人类不断建造更大可用系统的内在动力。

其中的大公司如下。

芯片供应商：Intel、Broadcom、Marvel、PMC 等。

设备提供商：Cisco、Juniper、华为、中兴等。

网络运营商：中国移动、中国联通、中国教育科研网、长城网（军方）Sprint、AT&T Verizon、Comcast。

2.7　计算机网络服务产业

类似于传统工业的制造业、零售业和服务业。依托计算机网络构建的基础框架设施，面向最终用户的多方面的计算机网络服务产业。面向最终用户的服务类大公司如下。

搜索：Google、百度。

门户：CNN 、Sina、NetEase。

即时通信：AOL、Skype、QQ。

电子商务：eBay、DangDang。

社区网络：Facebook、Blog 。

网络游戏：盛大、网易、九城。

面向最终用户的服务类（灵活的中小规模公司），通过定制系统、定制服务和紧密结合需求。

2.8　IT　产　业

IT 产业一般分为计算机工业和通信工业，计算机工业一般指计算机系统产业计算机终端、计算机网络产业和计算机网络服务产业。通信工业包括无线电信网络、传统固化网络和电信通信终端等产业。

计算机系统硬件类：Intel、AMD、Apple、Lenovo 等。

计算机系统软件类：Microsoft、Oracle 等。

计算机网络设备类 ：Cisco、Juniper、3Com、HuaWei、ZTE 等。

计算机网络服务类：Google、Yahoo!、Facebook、Baidu 等。

通信工业：Alcatel-Lucent、Nokia-Siemens、Ericsson、HuaWei、ZTE 等。

2.9　信息通信安全

通信安全是指在通信场景下，保护交互双方的信息交互的机密性、完整性、可认证性、非抵赖性和服务的可靠性。

举一个通信场景：假设 Alice（A）想和她的存款银行 Bank（B）进行一些金融事务过程。

1. 通信的机密性（Confidentiality）

不应出现未经授权的 Alice 信息的公开，除了 Alice(A)和银行 Bank(B)，其他任何人不应该获悉他们之间的信息交换。

2. 通信的完整性（Integrity）

不应出现未经授权的信息操作，除了 Alice(A)和银行 Bank(B)，其他任何人不能修改他们之间的信息交换。

3. 通信的可认证性(Authentication)

接收者 Bank（B)能确认信息的起源的能力。入侵者不能伪装其他用户，如 Alice（A)。

4. 通信的非抵赖性(Nonrepudiation)

接收者 Bank（B)应该能确认信息确实是 Alice（A)发出的能力。发送者 Alice（A)不能否认之前她发过的数据。

5. 服务的可靠性(Service Reliability)

应该具有保护通信会话抵抗拒绝服务攻击的能力。

如何满足信息通信安全需要，特别是信息安全的机密性、完整性和可认证性(CIA 特性)，需要密码学的知识与工具。密码学是理论计算机科学的重要内容。2002 年美国计算机学会 ACM 的图灵奖颁给了麻省理工学院(MIT)的 Ronald L. Rivest Adi Shamir 和 Leonard M. Adleman，以表彰他们提出了 RSA 公钥密码算法。2012 年美国计算机学会 ACM 的图灵奖又颁给了麻省理工学院的 Shafi Goldwasser 和 Silvio Micali 教授，以表彰他们对基于复杂理论的密码学的数学可证明性的贡献。

密码学主要研究计算数学、加解密算法、密码 Hash 函数、签名与验证、密码协议等。在这些基本密码工具基础上，架构对称密码体制、公钥密码体制、安全散列函数和密钥管理与分发。在此基础上，最终标准化形成了现在网络环境下的信任机制 PKI 框架和 X. 509 证书系统。基于 PKI 框架和 X. 509 证书是目前政府和商业网站防止被钓鱼和假冒的基本技术手段，也是互联网内容安全的基础。图 2-8 给出了美国国家标准与技术局(National Institute of Standards and Technology，NIST)制订的密码学标准及其分类。

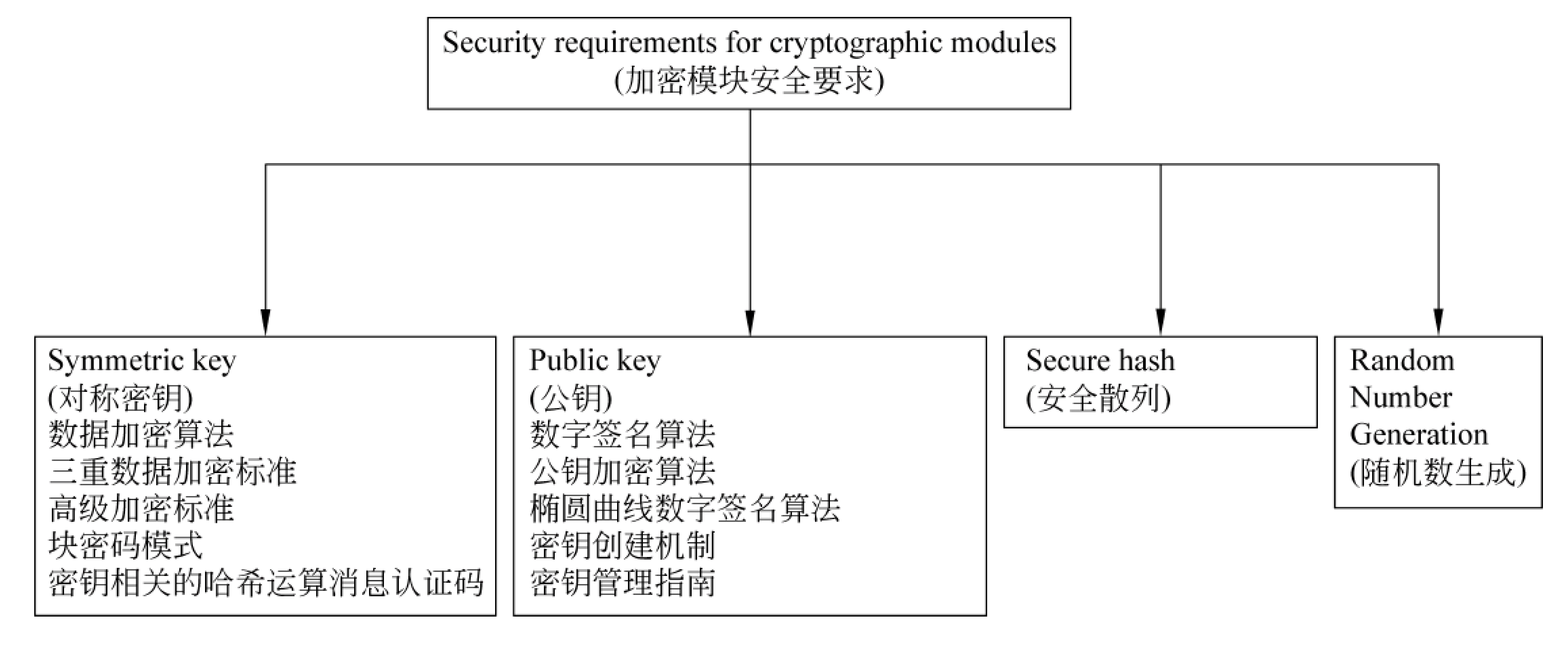


图 2-8　密码学基本分类及其标准(NIST)

我国的密码学标准主要是 SM2 椭圆曲线密码算法、SM3 密码杂凑函数和 SM4 分组加

密算法。SM2 椭圆曲线密码算法包括三个子算法：椭圆曲线数字签名算法（SM2-1）、椭圆曲线密钥交换协议（SM2-2）、椭圆曲线公钥加密算法（SM2-3）。

中国在密码学方面也做出了不少贡献，例如，上海交通大学的来学嘉教授和苏黎世联邦理工学院的梅西教授设计的 IDEA 加密算法，至今被 PGP（Pretty Good Privacy）使用。2004 年山东大学的王小云教授破解了 MD5 散列函数，接着又攻破了 SHA-1 散列函数，从而推动美国国家标准技术局 NIST 于 2008 年开始，进行新一轮的散列函数竞赛，以代替 MD5 和 SHA-1 函数。2012 年 10 月，NIST 正式宣布 Keccak 算法为 SHA-3 标准，Keccak 算法为比利时的研究组提出。对 SHA-3 的破解又将成为研究的一个新热点。

2.10　信息安全工业

信息安全工业包括计算机安全工业，计算机安全工业包括主机安全和网络安全，目前呈现两者的融合趋势。信息安全产品及服务如图 2-9 所示。

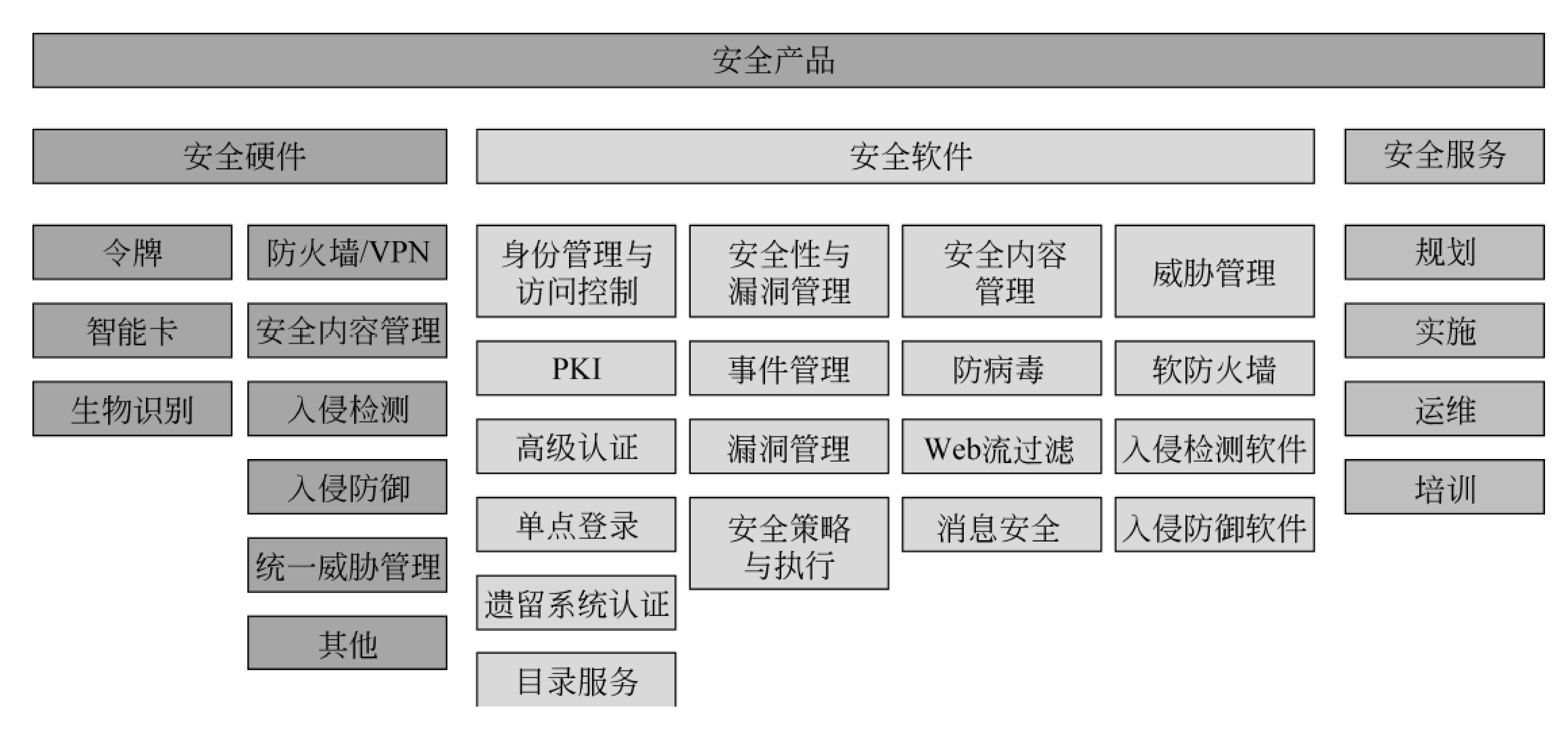


图 2-9　信息安全产品及服务

计算机安全工业是一个小众市场，多姿多彩。由于这个产业的特殊性，大家可能不太了解。安全工业是一个“隐蔽”的工业，造成这个行业的生态链具有比较封闭的特殊性。国家机构中负责安全工业监管的部门有公安部、安全部、中央机要局和国家密码管理局、总参通信部。

此外，每个行业都有各自的安全产业、安全标准和服务，如金融业、电信业等。

2.11　网 络 安 全

互联网安全是信息通信安全的一部分，即采用网络设备或者软件，提供对计算机系统和计算机网络的保护，抵抗可能的破坏和风险。互联网安全是计算机控制权的攻防，是计算机网络世界中的较量，带有浓厚的“军备竞赛”（Arm Race）特征。

网络安全的特点具有攻防两面：攻击的方法都是细致入微，针对非常细致的漏洞，属于

案例研究;攻击防护比攻击要困难,处于“易攻难守”境况,这和网络环境相关。

掌握网络知识是互联网安全的有效保障,知道攻击的知识和防守的办法,才会有良好的安全意识和物质技术保障。

2.12 互联网安全产业

互联网安全产业是IT工业的重要组成部分,自从互联网高速发展以来,逐渐被人关注(1993年至今)。

互联网安全研究是计算机网络学科的分支,具有很强的实践性和工程性。

互联网安全代表性的产品防火墙就是这个时候提出来的。

第 3 章　网络安全问题的产生

本章探讨互联网安全为什么会有这么多问题，其中互联网体系架构和软件系统脆弱性是网络安全一内一外的两大重要原因。

3.1　互联网的巨大成功——必要条件

基于 TCP/IP 技术的互联网获得了巨大成功，战胜了通信工业提出的 ATM\X.25 等其他网络技术方案成为事实的计算机网络标准，成为人类社会的基础信息网络框架。同时，WWW 的兴起为互联网的内容提供了一种标准的媒体描述方式，使得互联网不仅作为计算机网络而存在，同时具有丰富的媒体内容资源，促进了互联网爆炸式的增长。在此基础上，随着信息化的浪潮，人类社会的社会行为纷纷迁移到了互联网上，形成了电子政务、电子商务、电子金融、电子教学等，使得互联网的使用成为人类的生活习惯，互联网安全问题随之受到大家的关注，以致成为热点问题。

归结互联网的成功(参考互联网基本原理章节)，互联网技术作为传统电信网络的颠覆者，是一个开放的平台，从技术发展角度去考虑设计问题，提供了丰富的媒体内容，顺应用户的需求，顺应新事物由弱到强、由小到大的发展过程。主要因素如下。

(1) 从不可靠的网络出发，构建可靠的网络。

(2) 从互信、未考虑安全因素出发，构建安全网络。

(3) 从对等角度，构建大规模系统。

表 3-1 给出了互联网与电信网络的对比说明。

表 3-1　电信网络与 Internet 的比较

	电 信 网 络	Internet
设计用途	话音传送为主	数据传输网络
服务内容	电话、传真、短信	WWW Web、多媒体等
设计目的	商业化、可运营化	健壮、抗毁、自愈
商业模式	收取通信费用	初期无商业化、多元化
设计要求	保证通信质量	尽力传送，保证互连互通
结构模式	同质，从上到下，集中管理	异质、对等、分布式、自治
发展模式	集中式架构，集中规范，集中升级	代议制/工程师联盟/IETF 管理架构和方式
网络终端	固定通话，终端地址	计算机
设计理念	网络设计复杂，终端功能简单	网络简单，尽量把复杂功能留给终端

是否需要重新设计 Internet？答案是肯定的，也就是设计下一代互联网或者未来网络。

目前 IPv6 应用越来越广阔，事实上 IPv6 只是新一代互联网的一种而已。

如何设计呢？这主要是架构与工程的问题，理论问题不多。事实上，从 2010 年开始，各国都兴起了新的网络设计热潮，美国国家自然科学基金（NSF）在 2010 年一次资助了 4 个未来的互联网研究项目，以解决当前互联网存在的网络安全、移动性、应用僵化和可管理性等问题。我国在未来互联网研究中投资巨大，目前，清华大学已经和 NDN 项目组进行合作，以期望在该领域有所斩获。

其中最新潮的解决思路称为信息中心网络（Information Centric Networking，ICN），强调信息内容的互连。这其中最引人注目的是命名数据网络 NDN/CCN（Named Data Networking/ Content Centric Networking），由加州大学洛杉矶分校（UCLA）计算机科学系张丽霞教授领导 12 个学校联合开发。

3.2 计算机系统的安全漏洞

当前随着计算机系统的功能越来越丰富，系统的规模越来越庞大，软硬件系统自身的健壮性由于系统的复杂性而降低，造成系统存在很多安全"漏洞"。在软件开发过程中，开发的复杂性通过编程语言、编程类库、编译工具和操作系统调用而大大增加了安全漏洞的产生。同时在商业化的竞争压力下，在系统的性能功能与安全性之间的平衡中，系统开发对安全功能的重视不够，因为安全的代价是以大大牺牲性能为代价。软件系统漏洞增长趋势如图 3-1所示。

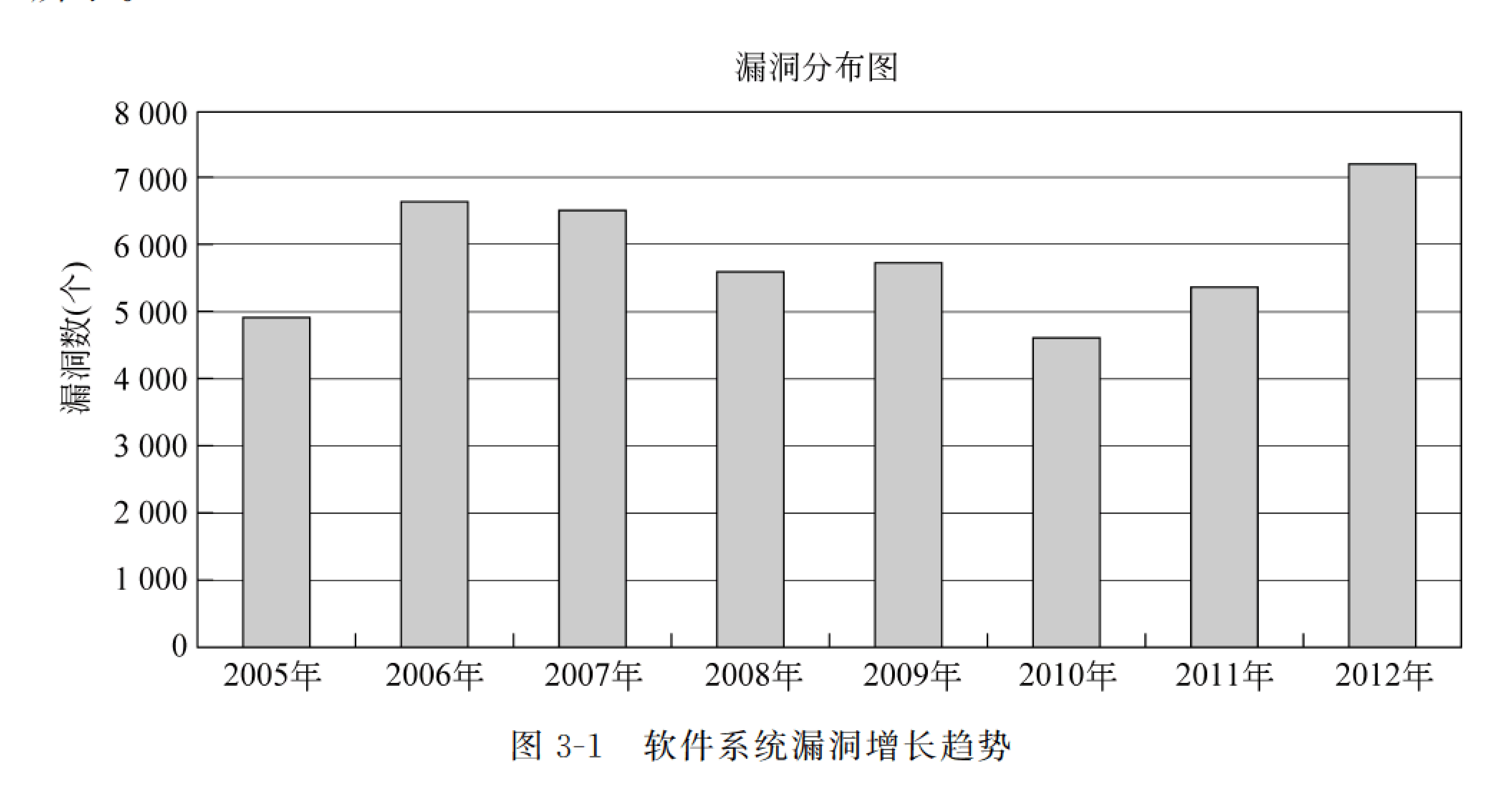


图 3-1　软件系统漏洞增长趋势

安全与用户友好性是一个比较大的矛盾：一个典型的案例是微软公司的 Windows XP，因其简单易用成为大众网吧的首选操作系统，虽然其安全性比较低，但易用性极好；相反，Windows Vista 系统吸收了 Windows XP 的安全问题，却因为重视安全性而导致易用性和性能的下降，大大减少了用户的接受程度。因此，微软公司不得不在 2009 年推出了 Windows 7，通过降低 Vista 系统的安全级别，以获得用户对其易用性的认可。从这个例子可以看出，互联网安全有其根本性的原因（人的因素）。

3.3 互联网的安全性

随着计算机网络技术的不断普及,尤其是互联网成为信息基础框架后,全世界的计算机采用 TCP/IP,通过路由器、交换机而连接起来,形成了全球互连互通的网络。这样,原本是单个计算机系统的安全漏洞通过网络的互连效应被大大地放大了。

更大的问题是,Internet 是一个分布式自组织的网络,其体系架构中缺乏统一的安全和管理框架。网络环境造成了攻击成本低,造成危害大的不利局面。比如动态 IP 地址分配、IP 路径重配置、NAT(网络地址转换)、P2P 覆盖网络等技术使得互联网安全事件的审计和追踪非常困难,安全攻击追踪受制于管理域问题而不能得到应有的“惩戒”。

3.4 互联网安全的“黑金”现象

目前很多病毒,如蠕虫(Worm)、木马(Torjan)和机器人网络(Botnet)的背后都有地下经济的黑影。互联网安全事件从单纯的“找乐”(For Fun)、显示“个人成就”等动机,迅速转化为受经济利益驱动的具有黑金性质的地下产业。制作病毒,传播病毒,控制机器人网络,散发兜售广告邮件(SPAM)成为一条完整的地下经济产业链。

加州大学伯克利分校的 Vern Paxson 教授研究表明,通过控制机器人网络来散发兜售广告邮件,是成本收益比非常低的销售手段,从而为机器人网络扩散提供了很强的经济动机。

此外,通过控制机器人网络,对政府、商业等网站进行分布式拒绝服务(Distributed Denial of Service,DDoS)攻击,表达政治诉求或者谋求经济利益,也是目前互联网安全事件频发的因素之一。

第 4 章　互联网的基本原理

前面提到了互联网的安全性问题,为了进一步说明其深层的原因,下面对其基本原理进行简要介绍。

4.1　互联网的定义

解决人人之间的通信问题,最基本的手段是给每对人之间都拉上专线光纤,形成点对点网络,如图 4-1 所示。但是因为经济成本的问题,这样做是不可行的,所以需要设计一个接入网络,人人先接入局部网络(或叫局域网,对于世界范围,局部也是相对的),然后由路由器(Router)将这些小网络连接起来,形成更大的世界范围的网络,这就是互联网的端到端的基本思想(End-to-End),如图 4-2 所示。



图 4-1　点对点网络



图 4-2　端到端的跨网络相连

实际的互联网拓扑图如图 4-3 所示。

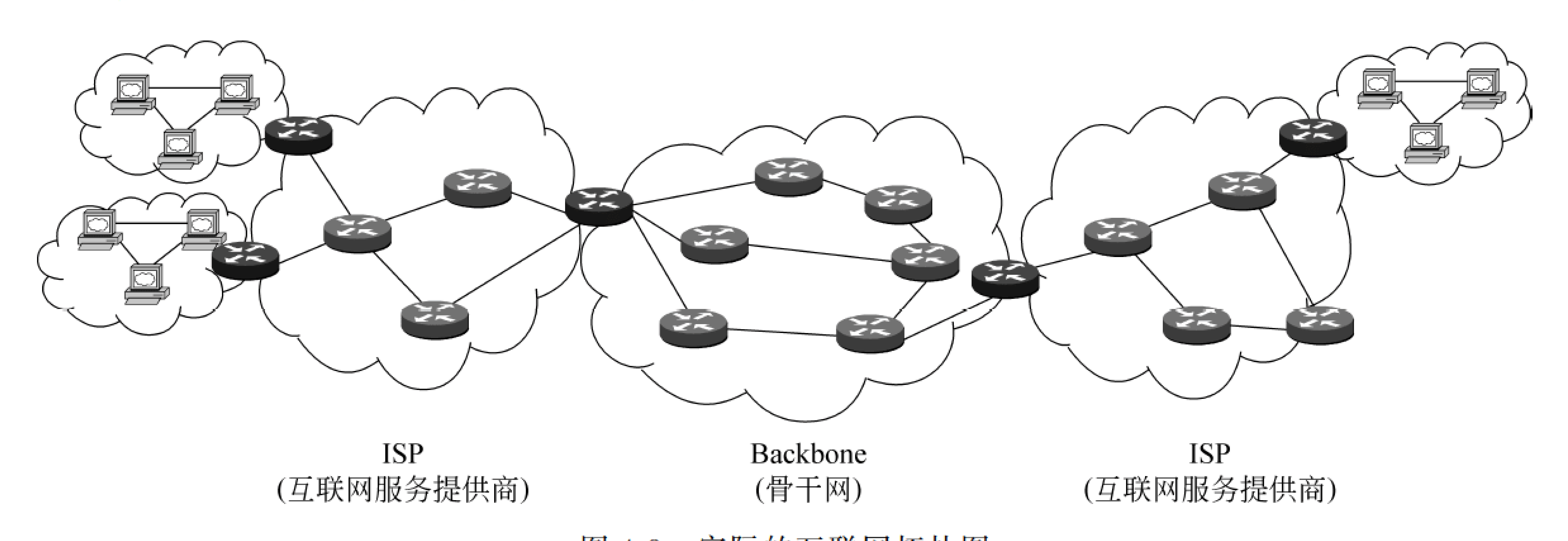


图 4-3　实际的互联网拓扑图

互联网采用 TCP/IP,由 Vinton G. Cerf 和 Robert E. Kahn 于 20 世纪 70 年代设计,并于 1980 年 9 月公布为 IETF RFC 791 标准。TCP/IP 为不同的数据网络(局部网络)提供了一套标准的协议,使得不同的数据网络按照这一套标准语义和语法,能够建立路由和通信。为此,TCP/IP 设计者获得了 2004 年的 ACM 图灵奖。

互联网的设计与人类社会运行几千年的邮政系统是一致的。正如书面信函一样,只要将信函写好,装上信封,写上收信人地址和寄信人地址,投入邮局的邮箱就可以了。互联网采用 TCP/IP 一样的运作模式。

首先,互联网采用 IP 协议通过编址为全世界的计算机都分配一个 IP 地址(固定地址,类比于收信人和寄信人地址),IP 地址中包含了该计算机所在区域号(即 IP 地址的网络地址,Network Address,类比于邮政编码 Postal Code),以及在该区域内的详细地址,即主机地址(Host Address)。

其次,用户要发送的消息分解成许多数据包(又叫做网包,Packet,类比于一封封信函),送给的第一个路由器叫网关(也叫默认网关,Default Gateway,类比于邮局邮箱),路由器先本地保存这份数据,然后等待机会发送到下一跳的路由器上,这样一跳一跳地路由(这种模式叫做存储转发,类比于邮政职工将信件从一个邮局送往下一个邮局)。

如何确定信件的下一个路由器需要 IP 网络的路由系统(Routing System)来确定。路由系统确定了从 A 点到 B 点路由过程中的传输路径,它是通过在每个路由器上维护一张路由表来实现。每个路由器独立维护自己的路由表。每个路由器都会主动向邻居路由器通报自己的路由表信息,根据相邻的路由器的路由表来更新自己的路由表。

当一个网包到达之后,通过查找路由表信息,就可以将这份数据传送到下一个路由器上,最终到达目的计算机上,在目的计算机上进行数据重组,呈现给用户。

这样就完成了信息从 A 点计算机转移到 B 点计算机的过程。

4.2 互联网的人机接口——域名系统

互联网通过所有计算机遵守 IP 协议完成了信息在 A 点计算机和 B 点计算机之间的数据传送。

这时专门用于提供信息内容的计算机出现了。但是信息的内容如何显示与创建、如何发布需要规范,这就是 W3C 的 WWW 的标准。浏览器是显示这些标准格式内容的软件。

随着浏览器的普及,以 W3C 标准的 HTML 的 Web 内容越来越多,这时提供这些 Web 内容的网站也越来越多,如 Google、Sina、Sohu 等门户网站。怎么搜索定位这些内容的计算机就成了问题,如何将 sina.com 映射成 IP 地址,这就有了域名系统(Domain System),而存储这些域名与 IP 地址映射关系的机器就是域名服务器。

域名系统是互联网的人机接口,为每个提供内容的计算机确定一个可读的名字。互联网的内容通过域名系统就能够精确定位,从而形成了一个提供内容和服务的互联网,这才是互联网的最重要的用处,也就是“上网”的意义。

4.3 互联网的交通运输——路由系统

IP 网络的每个网包因为具有独立的发信人地址和寄信人地址,因此路由器对这样的网包进行独立路由选择,互联网的信息选路和信息传送是同步进行的。

首先,IP 网络中的每个遵循 IP 协议的机器都是一个路由器,根据路由表来转发数据,如果路由表项的下一条是自己,则说明自己是这个信息的接收者,否则的话,就尽力把信息传送给其他路由器,如默认网关路由器或者其他相邻路由器。

互联网的路由系统也是分布式的,按照网络管理域分为域间路由和域内路由。分别采用不同的路由协议来更新每个路由器的路由表。

对于用户而言，互联网的路由系统是不可见的，用户只要配置好自己的默认路由器就可以实现网络服务的接入，因此是没有网络接入访问控制的。

4.4 IP 协议

IP 协议尽力而为传送数据到达指定地址，不确认数据是否正确到达，是一种无连接(Connectionless)的协议。

1. IP 地址

当前 IP 协议版本 4(Internet Protocol Version 4，IPv4)采用 IP 地址辨别互联网上设备或主机，是唯一的识别码，具有全局效应，适用于不同网络间寻址。

IP 地址由 32b 组成，理论上讲可有 $2^{32} = 4\ 294\ 967\ 296$ 个地址。

IP 地址采用分层结构，由网络地址(Network Address)和主机地址(Host Address)两部分组成。依照网络地址和主机地址的分配，分成 5 类地址，即 A 类、B 类、C 类、D 类和 E 类地址，如图 4-4 所示。分成 5 类地址主要原因是网络的规模(内部的计算机数目)多种多样。

	1st字节	2nd字节	3rd字节	4th字节	主机数
类A	0 网络地址	主机地址			16 777 216
类B	10 网络地址		主机地址		65 536
类C	110 网络地址			主机地址	256
类D	111 多播				多播
类E	1111 保留				保留

图 4-4 IP 地址的分类

清华大学的 IP 地址为 59.66.0.1/16(A 类)和 166.111.0.1/16(B 类)。采用 ipconfig 命令可以有效确定 IP 地址，信息如下。

```
C:\Users\zhenchen> ipconfig
Windows IP 配置
以太网适配器 Local Area Connection:
   连接特定的 DNS 后缀 . . . . . . . : tsinghua.edu.cn
   IPv4 地址 . . . . . . . . . . . . : 166.111.137.197
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 166.111.137.1
```

2. 子网掩码

如何获取 IP 地址的网络地址呢？这个问题由子网掩码(Subnet Mask)来解决，如图 4-5 所示。

3. 分割子网

将一较大的网络区段切割成几组较小的网络，使用于内部路由选择协议(Interior Routing Protocol)进行路由交换。

例如，B 级网络 59.66.0.0/16 可拆分成 256 个较小网络。

```
IP 地址     11000000  10101000  00000001  01110100
            |◄──────────── 27位 ────────────►|
子网掩码    11111111  11111111  11111111  11100000
  AND
网络号      11000000  10101000  00000001  01100000
               192   ·   168   ·    1    ·    96
```

图 4-5　子网掩码

```
59.66.0.0 / 24
59.66.1.0 / 24
59.66.2.0 / 24
⋮
59.66.255.0 / 24
```

4. IP 协议数据格式

IP 协议头就是普通书面信函的信封，其格式如图 4-6 所示。信封内装的信纸，就是这里的数据。

0	4	8	12	16	20	24	28	32
4b 版本号	4b 头部长度	8b 服务类型		16b 报文总长度(B)				
16b 标识				3b 标志	13b 片偏移量			
8b 生存时间		8b 协议		16b 首部校验				
32b 源IP地址								
32b 目的IP地址								
选项								
数据								

图 4-6　IPv4 协议头

为什么寄信人地址要写在收信人之前呢？因为 IP 协议是美国人设计的，大家可以注意一下西方的普通书信格式是寄信人在前，收信人在后，体现对收信人的礼貌，因此这种设计就是文化传统的问题，如果让中国人来设计的话，可能就会把这两项给颠倒过来。

4.5　ICMP 协议

互联网的运作离不开故障的管理，ICMP（Internet Control Message Protocol，互联网控制消息协议）设计用来处理网络设备之间的错误，报告网络环境中错误状态的发生，但是无法确保能把消息确实送达或是转回发送地。

ICMP 的一个应用是 Ping，用来测试机器之间的连通性，如下例所示。

```
C:\Users\zhenchen> ping www.tsinghua.edu.cn
正在 Ping www.d.tsinghua.edu.cn [2001:da8:200:200::4:100] 从 2001:da8:200:900e:2
```

```
00:5efe:166.111.137.197 具有 32 字节的数据:
来自 2001:da8:200:200::4:100 的回复:时间<1ms
来自 2001:da8:200:200::4:100 的回复:时间<1ms
来自 2001:da8:200:200::4:100 的回复:时间<1ms
来自 2001:da8:200:200::4:100 的回复:时间<1ms
2001:da8:200:200::4:100 的 Ping 统计信息:
  数据包:已发送=4,已接收=4,丢失=0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
  最短=0ms,最长=0ms,平均=0ms
```

ICMP 的另外应用是 traceroute(UNIX 下)和 tracert(Windows 下),如下面的 tracert 一例。

```
C:\Users\zhenchen> tracert www.berkeley.edu
通过最多 30 个跃点跟踪
到 www.w3.berkeley.edu [169.229.131.81] 的路由:
1      <1 ms     <1 ms     <1 ms   SECURITY [192.168.128.1]
2       1 ms     <1 ms     <1 ms   166.111.137.1
3      <1 ms     <1 ms     <1 ms   tu128098.ip.tsinghua.edu.cn [166.111.128.98]
4       1 ms     <1 ms     <1 ms   tu128101.ip.tsinghua.edu.cn [166.111.128.101]
5       1 ms      1 ms     <1 ms   th004133.ip.tsinghua.edu.cn [59.66.4.133]
6       4 ms      1 ms      1 ms   118.229.2.10
7       1 ms      1 ms      1 ms   118.229.2.14
8       1 ms      1 ms      1 ms   118.229.2.2
9      10 ms     11 ms     11 ms   th002237.ip.tsinghua.edu.cn [59.66.2.237]
10     11 ms     11 ms     10 ms   pku0.cernet.net [202.112.38.73]
11     11 ms     11 ms     11 ms   202.112.53.169
12      3 ms      3 ms      3 ms   202.112.61.158
13     12 ms     11 ms     11 ms   202.112.53.18
14    100 ms    100 ms    100 ms   tpr5-ae0-25.jp.apan.net [203.181.194.125]
15    215 ms    214 ms    215 ms   losa-tokyo-tp2.transpac2.net [192.203.116.145]
16    206 ms    205 ms    206 ms   cenichpr-1-lo-jmb-702.lsanca.pacificwave.net
                                   [207.231.240.129]
17    214 ms    213 ms    214 ms   svl-hpr--lax-hpr-10ge.cenic.net [137.164.25.13]
18    215 ms    215 ms    215 ms   oak-hpr--svl-hpr-10ge.cenic.net [137.164.25.9]
19    217 ms    215 ms    216 ms   hpr-ucb-ge--oak-hpr.cenic.net [137.164.27.130]
20    225 ms    225 ms    225 ms   t2-3.inr-202-reccev.Berkeley.EDU [128.32.0.39]
21    222 ms    217 ms    216 ms   t1-1.inr-211-srb.Berkeley.EDU [128.32.255.43]
22    216 ms    217 ms    216 ms   webfarm.Berkeley.EDU [169.229.131.81]
跟踪完成。
```

4.6 应用协议端口 Port

IP 使得互联网上的任何两台计算机之间能够建立尽力而为的通信,但是当计算机上的多个通信程序同时使用网络时,还需要在 IP 上设计数据传输协议,并引入端口的概念来区

分不同通信应用程序。

IP 网络类似于邮政系统，如在信件传递的基础上，邮政系统又推出了快件、加急快件服务(如 EMS)、平信和挂号信等服务。类比于互联网，互联网的 TCP 和 UDP 就是在普通信件服务上增加的这些专用的服务，以保证不同业务的服务要求。

IP 之上的数据传输协议主要包括 UDP 和 TCP。应用程序在互相通信时需要选择数据传输协议 UDP 或者 TCP，以及相应的传输端口，以区别其他应用程序，如图 4-7 所示。

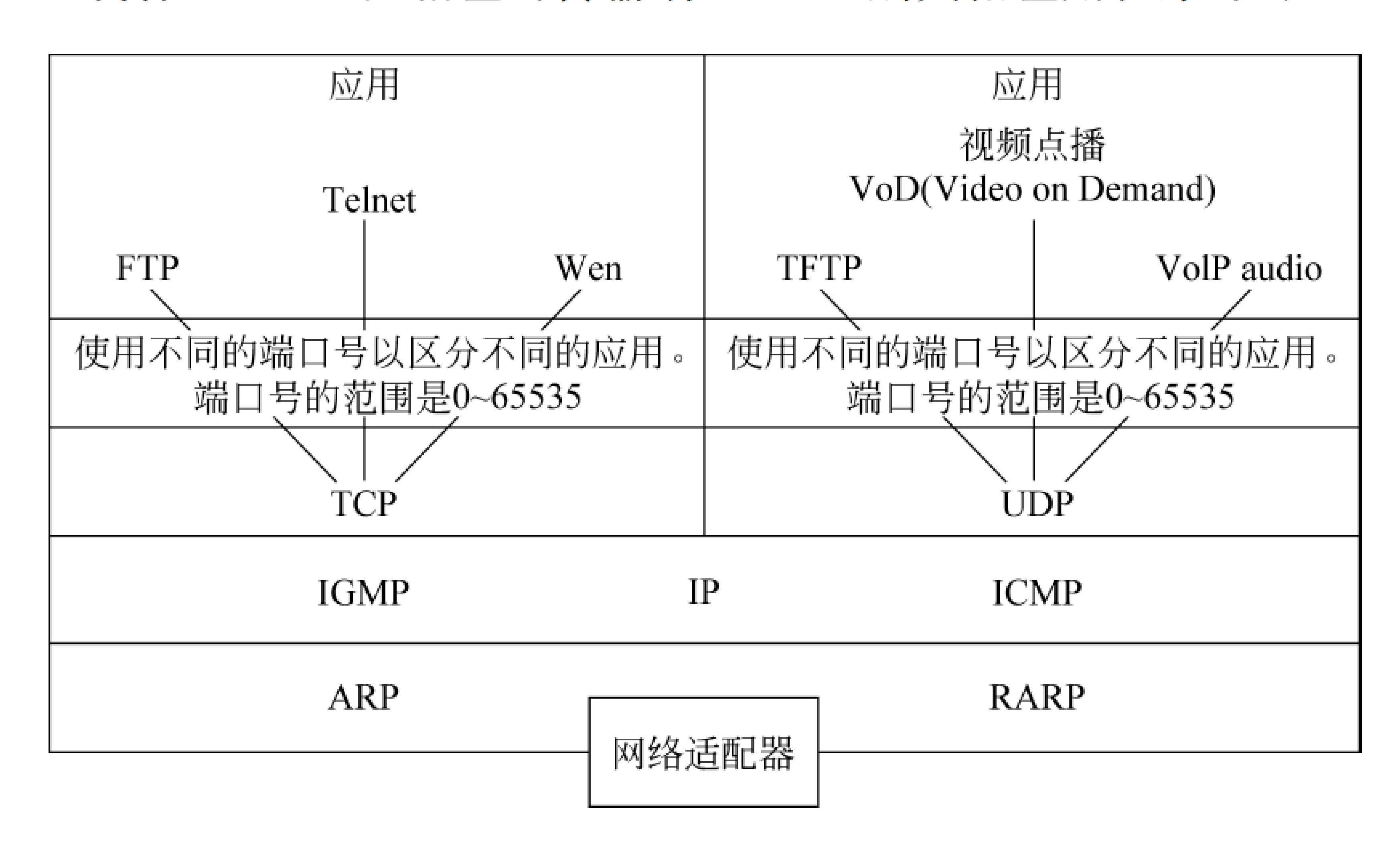


图 4-7　TCP/IP 协议栈

4.7　UDP

讲究传输速度且允许一点数据丢失(Data Loss)或是数据包丢失(Packet Loss)的常用传输协议是 UDP(User Datagram Protocol)。例如 GTalk、MSN、Skype、QQ 等使用语音通信的应用程序均使用了 UDP。

UDP 定义于 RFC 768，提供应用程序能够在最低的协议机制下发送消息给其他应用程序，但不保证是否可靠或有没有依照传送顺序到达。

UDP 数据格式如图 4-8 所示。



图 4-8　UDP 数据格式

4.8　TCP

IP 网络偶尔存在丢包的情况，为了在这种情况下还能有效保障传输的质量，需使用 TCP，如文件传输服务。例如，从 FTP 服务器下载电影需要将一个 1GB 大的电影文件分成很多网包在网络上传输，如果其中一份数据丢失将导致整个文件无法打开(因为校验不通过)。

为了解决可靠传输的问题，TCP 的基本思想是为每次通信会话保留状态，为传输的每段数据编号，通过发送和接收端的互相通报以及保存的状态信息来确定可能的网包丢失情况，通过重新传输丢弃的网包来保证可靠性。

TCP 提供以下三个基本功能。

（1）可靠性（Reliability）：克服包丢失现象，传送的数据依照顺序交给程序。

（2）复用性（Multiplexing）：通过不同的端口可使同一个 IP 地址（一台计算机）可以同时提供上层不同的多种服务，如 FTP、Telnet、HTTP Web 服务等。

（3）流控（Flow Control 或 Congestion Avoidance and Control）：避免网络或接收端拥塞，以便提供高效率的传输。

TCP 的会话叫做连接（Connection），TCP 的会话建立需要三次握手，如图 4-9 所示，会话拆除也需要握手。这种握手协议也是 TCP SYN 攻击的原因，一种异常的 TCP SYN 泛洪攻击如图 4-10 所示。

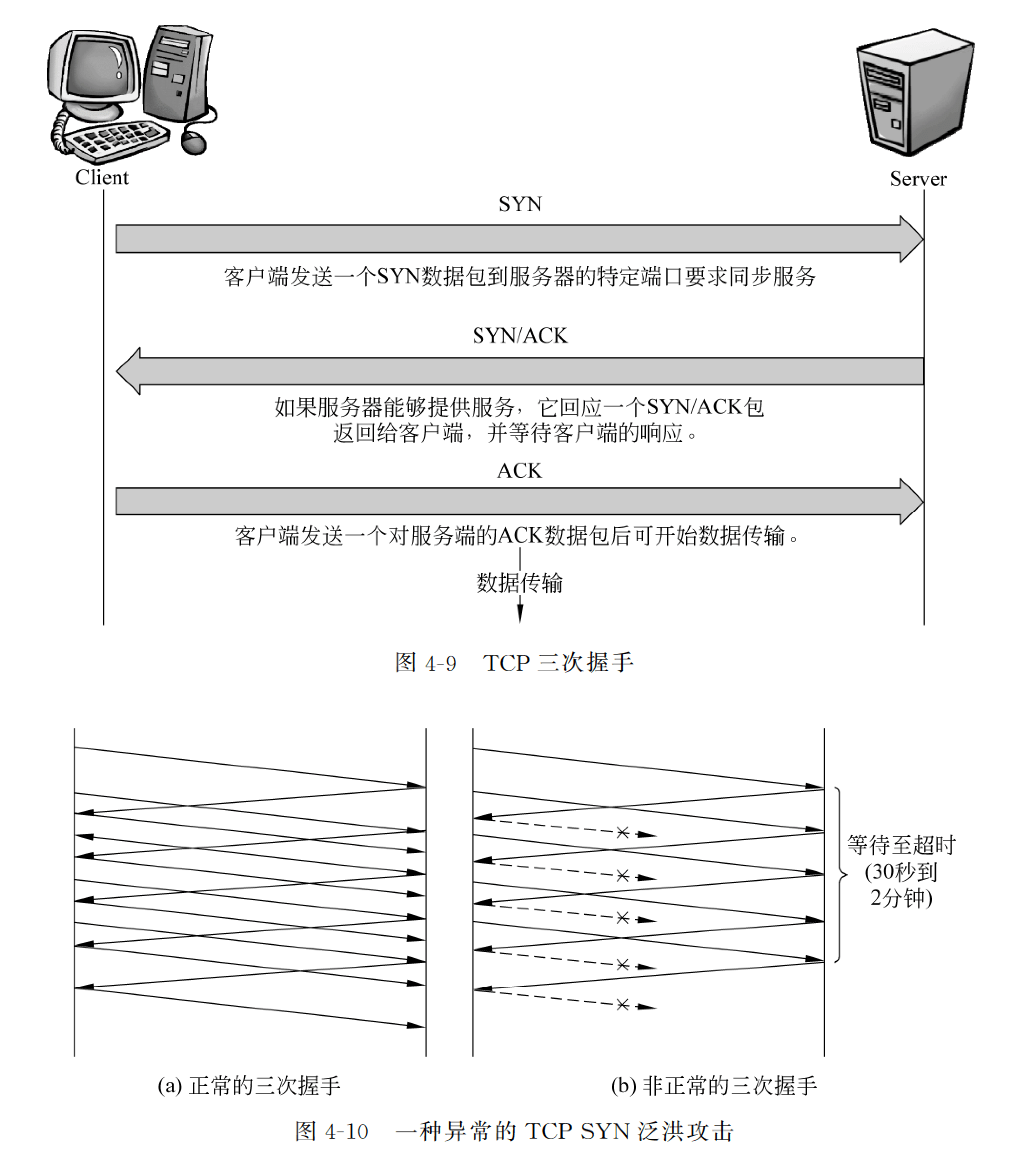


图 4-9　TCP 三次握手

图 4-10　一种异常的 TCP SYN 泛洪攻击

TCP 的格式如图 4-11 所示。

0						15 16		31
16位源端口号								16位目的端口号
32位序列号								
32位确认序号								
数据偏移	保留	URG	ACK	PSH	RST	SYN	FIN	16位窗口尺寸
16位校验和								16位紧急指针
选项								填充位
数据								

图 4-11　TCP 的格式

4.9　以　太　网

1．以太网(Ethernet)简介

以太网是使用最广泛的局域网类型。10M/100M 以太网采用共享网络介质，属于共享带宽；吉比特以太网（Gigabit Ethernet）采用交换网络方式，属于独享型带宽。

2．以太网协议

以太网协议见 RFC 894，以太网协议帧格式如图 4-12 所示。

IEEE 802.2/802.3 (RFC 1042)

802.3MAC			802.2LLC			802.2SNAP			
目的MAC	源MAC	Len	DSAP	SSAP	Control	Org Code	Ether Type	Payload	CRC
6	6	2	1	1	1	3	2	38-1492	4

Ethernet (RFC 894)

目的MAC	源MAC	Ether Type	Payload	CRC
6	6	2	38-1492	4

图 4-12　以太网协议帧格式

3．5 类以太网络线

以太网网线为 10Base-T 及 100Base-T，使用 8 芯（4 对线）的无遮蔽双绞线(Unshielded Twisted Pair，UTP)网络线，接头为 RJ-45，接法有平行接法和串接接法，具体连接关系如图 4-13 所示。

2, 6

568A：白 绿、绿、白 橙、蓝、白 蓝、橙、白 棕、棕

568B：白 橙、橙、白 绿、蓝、白 蓝、绿、白 棕、棕

1, 3

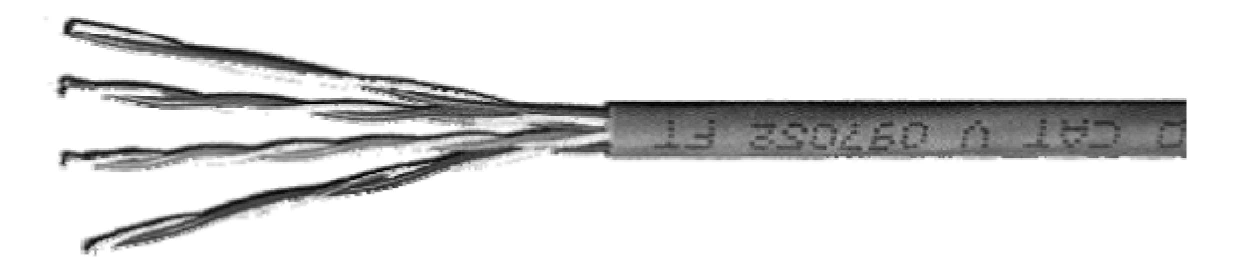


图 4-13　以太网 RJ-45 接口走线

第5章　安全威胁

计算机安全威胁主要有黑客攻击(Hacking)、网络欺诈(Phishing)、计算机恶意代码、机器人网络等内容。计算机网络安全防范有杀毒软件、防火墙等技术,下面分别介绍这些内容。

5.1　黑客攻击

黑客攻击是以获取计算机控制权为目的的网络攻击。整个过程犹如特种部队作战,因此有虚假的成就感。如目标信息采集,锁定目标,目标攻击,隐匿善后等。在目标信息的采集上,一般通过对网络和主机进行探测获取信息,或者由外入内,通过社会工程等方法获取信息。一旦锁定目标之后,即用各种安全工具(如 Metaploit 等),或者特洛伊木马、病毒和蠕虫等,利用各种漏洞,如邮件服务、文件服务和网页服务等安全漏洞,获取目标的一定权限然后再通过本地用户攻击等提升权限。入侵成功之后,隐藏后门,销毁痕迹,或者植入内核 Rootkit,以便以后进一步利用。

由于黑客攻击都比较隐秘,需要专门的安全流量审计来进行辅助分析,我们所研究的协同式网络安全防御系统,旨在通过分布式部署流量探针设备来发现可能的黑客攻击。

5.2　网络欺诈

网络欺诈或称为钓鱼攻击(Phishing Attack),是一种盗取用户银行或者电子商务账户的攻击方法,主要是通过构建假冒的网页或者网站,通过垃圾邮件的散播,来吸引用户上当,比如声称用户的账户密码不安全,需要重新修改,或者中奖信息,吸引用户点击。

网络欺诈流程图如图 5-1 所示。

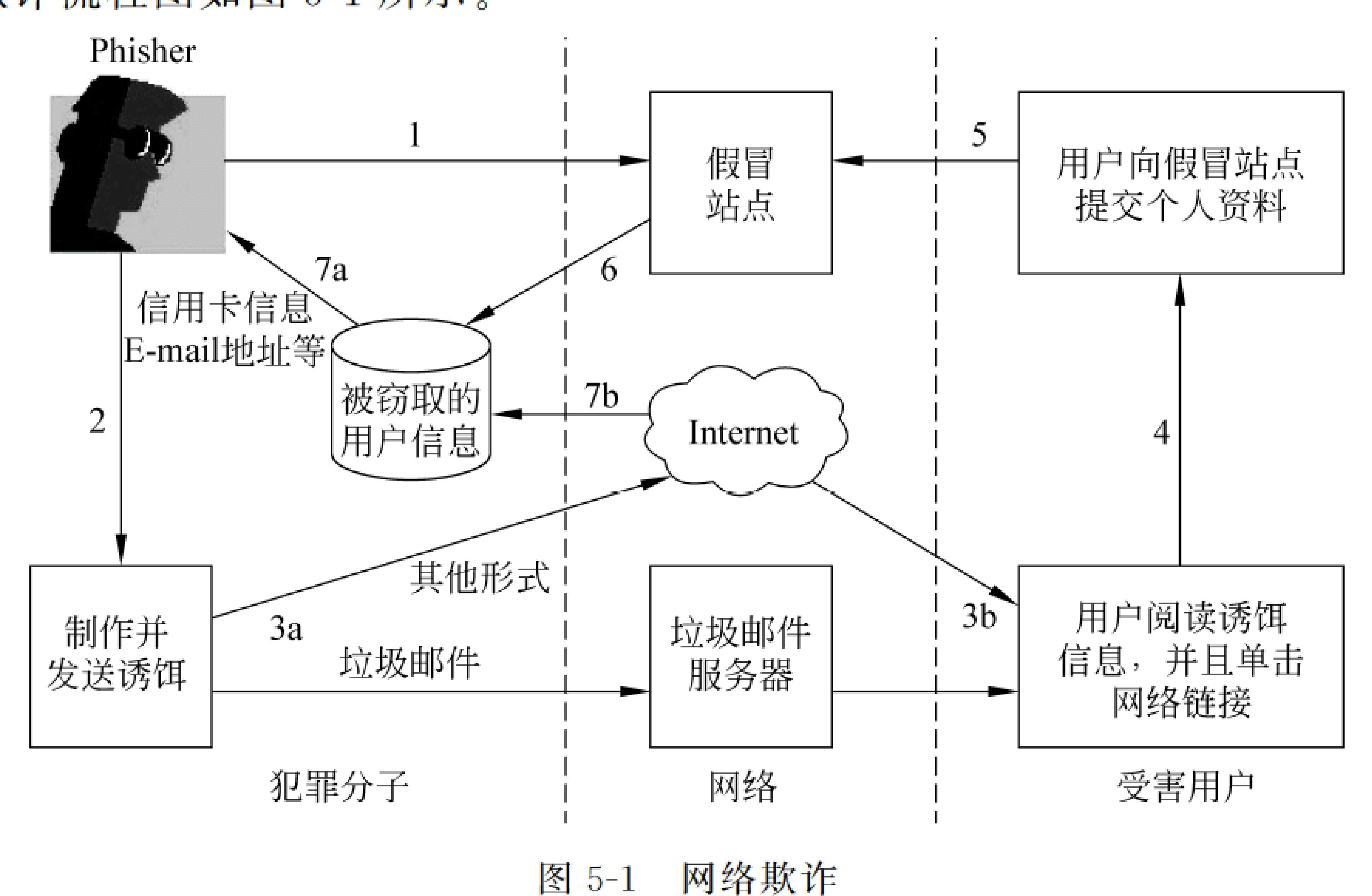


图 5-1　网络欺诈

网络欺诈目前已经形成了一条地下经济链条，如图 5-2 所示。因此各种攻击方法和手段日益纯熟。在这条经济链条上，有着不同的分工，如构建假冒的网页或者网站，制作机器人软件，控制机器人网络，控制垃圾邮件的散播，每一个环节将都从最终用户的损失中获得利益报酬。

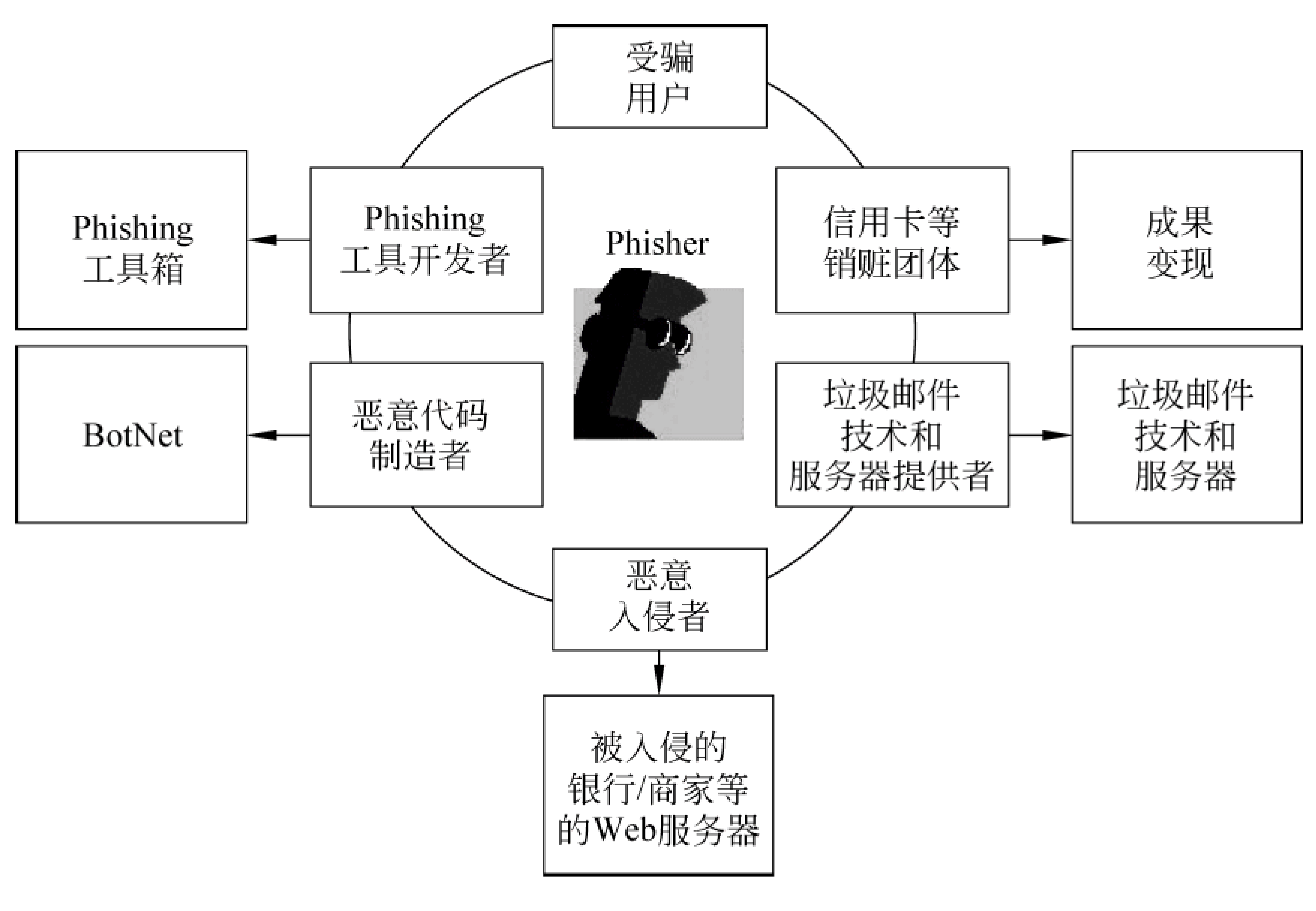


图 5-2　网络欺诈的经济链条

5.3　计算机恶意代码

中华人民共和国公安部第 51 号令《计算机病毒防治管理办法》于 2000 年 4 月 26 日颁布，其中计算机病毒(Virus)定义为：计算机病毒是指编制或者在计算机程序中插入的破坏计算机功能或者毁坏数据，影响计算机使用，并能自我复制的一组计算机指令或者程序代码。

更广义一些，计算机恶意代码(Malware)是指一个插入到系统中的程序(通常会故意隐匿)，意图破坏受害人的数据、应用程序或操作系统的完整性、保密性和可用性，干扰或中断用户的正常使用。

(1) 特洛伊木马是一种不能自我复制的程序，看似无害，实际上具有隐藏的恶意目的。

(2) 蠕虫病毒是完全自包含、自复制的一段程序，不需要宿主程序就可以感染目标对象。

(3) 移动代码是从远程系统传输到本地执行的软件，一般不需要用户的明确指令。移动代码不感染文件，也不会自我复制，所以从根本上有别于蠕虫病毒。不同于蠕虫病毒会利用特定的系统漏洞，移动代码常常利用赋予其自身的系统默认特权来感染系统。

(4) 间谍软件是一种未经许可或用户知晓、在系统后台窃听用户网络使用状况，并且收集或回传用户信息和用户行为信息的应用程序。

1. 特洛伊木马

在神话中，表面上特洛伊木马是“礼物”，实际上却是藏匿袭击特洛伊城的希腊士兵。

现在，特洛伊木马是一些表面有用的软件程序，实际目的是危害并破坏计算机安全。

木马实质是一个 Client/Server 程序，以远程访问控制受害机器，获取控制权和密码，以进行其他操作为目的，如冰河木马。

作为不合法的网络服务程序，木马必须隐藏自己。

木马的传播方式一般为非主动传播，属种植型。最近的特洛伊木马都以电子邮件的形式传播。

2. 蠕虫病毒

蠕虫病毒可自动完成复制过程，控制计算机中传输文件或信息的功能，一旦计算机感染蠕虫病毒，蠕虫即可独自传播，并大量复制。

蠕虫病毒传播的特点是依靠网络主动传输，攻击受害机器，传播迅速，消耗极大的网络和计算资源。

一个典型攻击 DNS BIND 服务器的蠕虫 Lion，利用 Bind 程序的缓存溢出作为突破口，攻击 Bind 服务器，控制 Bind 服务器利用 HTTP 协议下载病毒体，确定 B 段 IP 地址作为攻击对象，进行传播。蠕虫病毒形象示意图如图 5-3 所示。

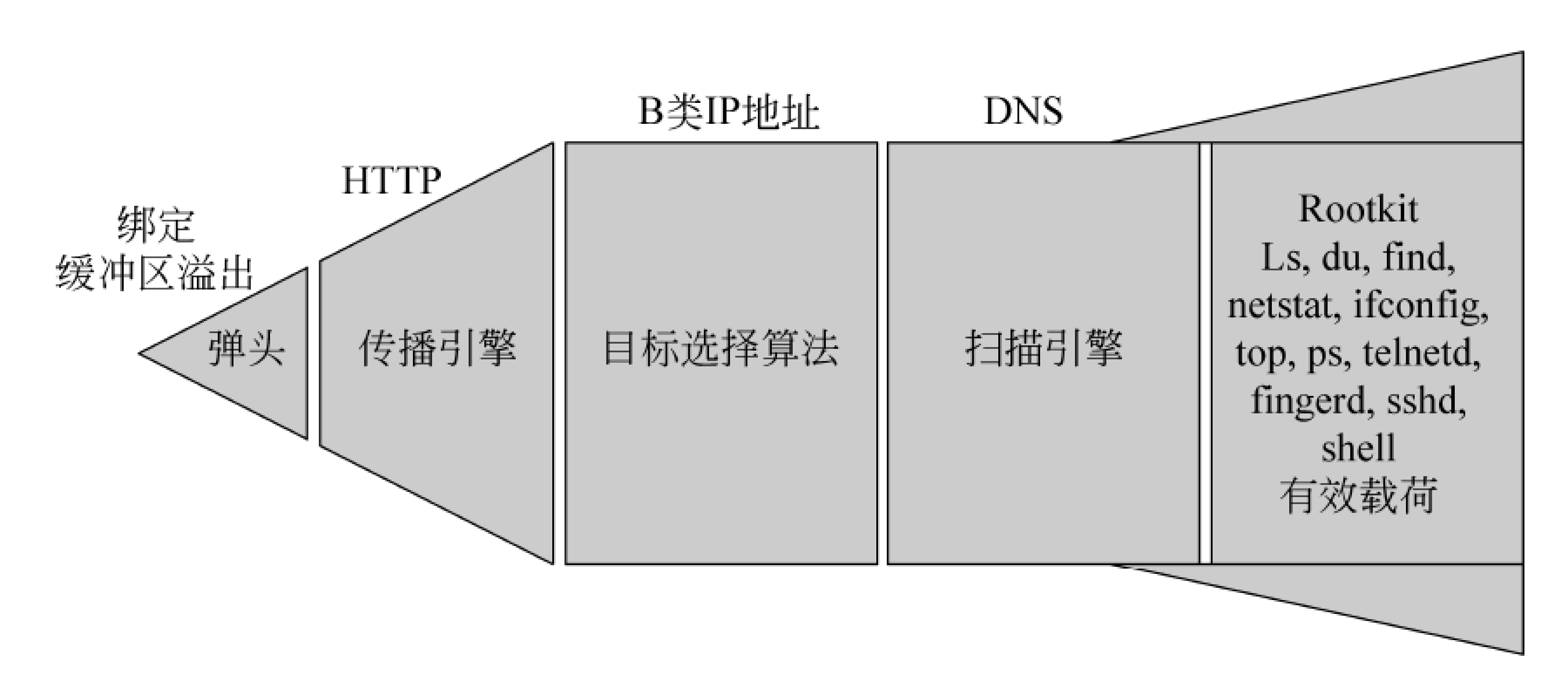


图 5-3　蠕虫病毒形象示意图

最为普遍的是红码病毒(CodeRed)，它具有多个不同版本，主要攻击 Windows XP IIS 系统漏洞，曾经风靡校园网络。

绝大多数蠕虫病毒主要是利用软件漏洞发布到漏洞补丁更新发布的时间差，从目前看，从软件漏洞发布到利用该漏洞的蠕虫出现的时间越来越多，近乎零日攻击(0day)。零日现象-软件漏洞公布日及其利用的病毒的时延如图 5-4 所示。

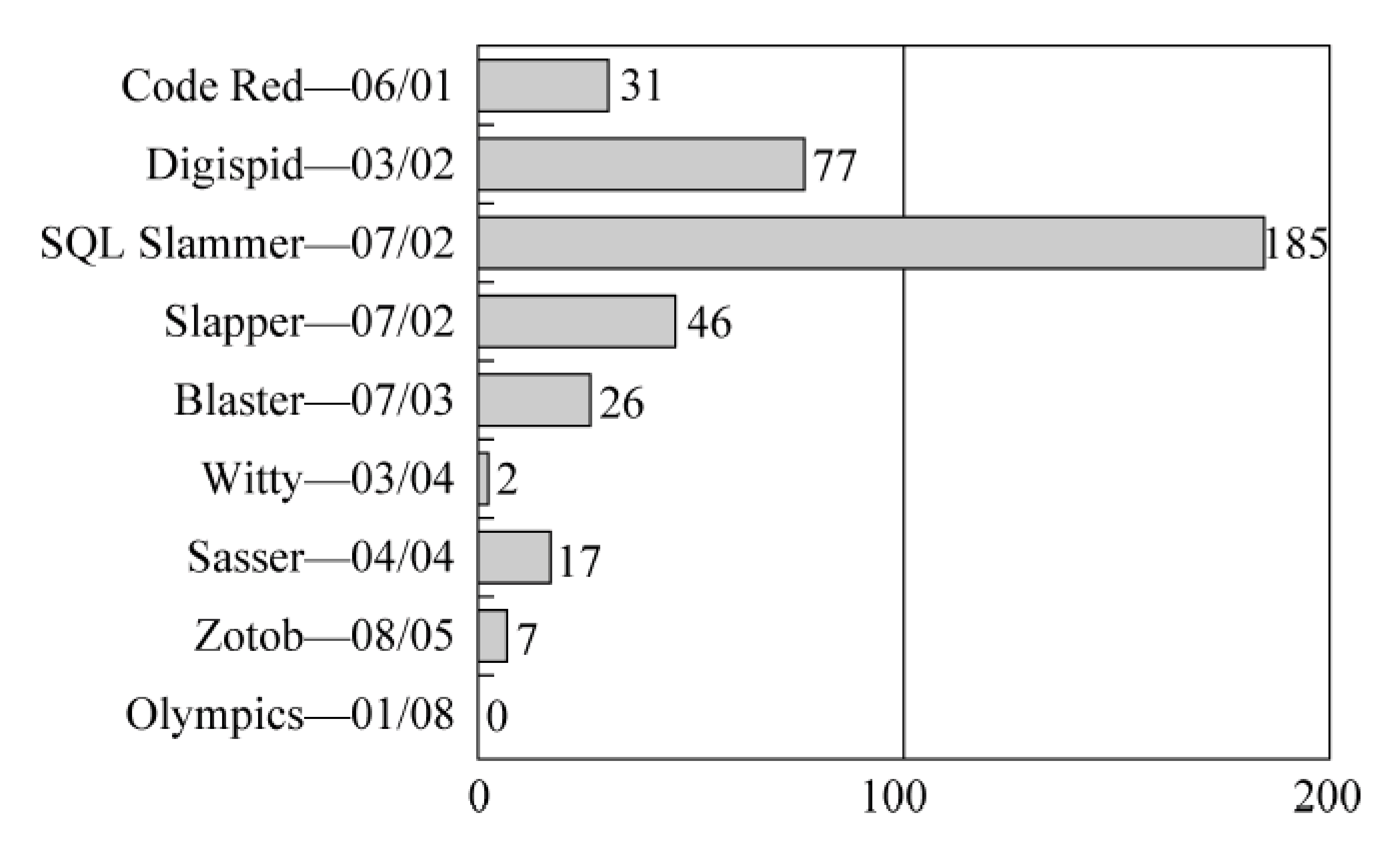


图 5-4　零日现象-软件漏洞公布日及其利用的病毒的时延

3. 恶意代码防范

恶意代码防范要参考传染病的防范模式,如图 5-5 所示。

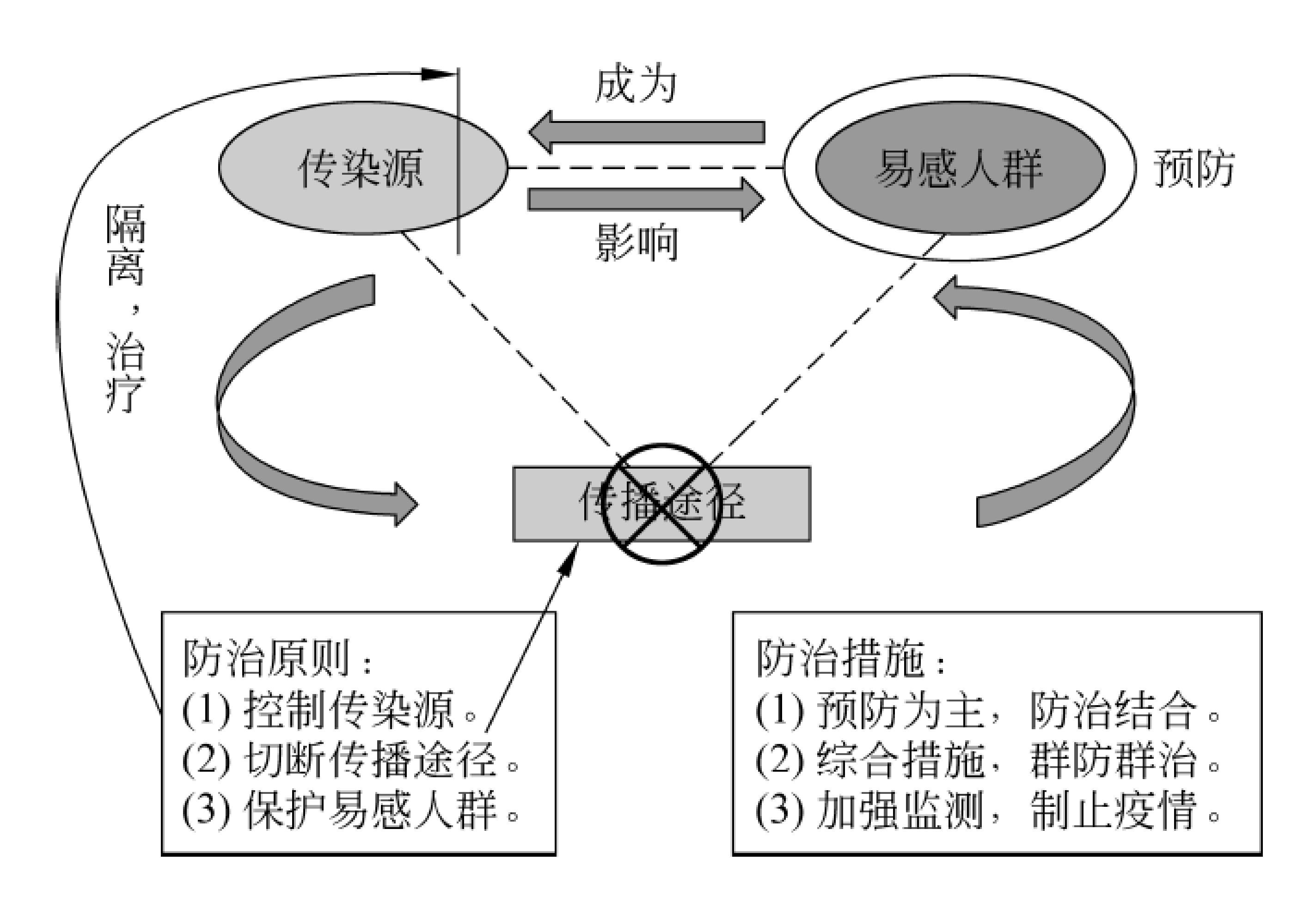


图 5-5　参考传染病防治的计算机病毒防范

5.4　机器人网络

Bot 程序是一种恶意代码,能够创建后门程序,创建代理程序,转发邮件,记录键盘输入,获取口令、信用卡号码及其他信息,个别还能停止个人防火墙、杀毒软件等安全软件。Bot 程序一般会加壳保护,自动升级到新版本。

植入 Bot 程序的计算机,在用户不知情的情况下已经被黑客控制,俗称"肉鸡"。黑客一般通过 IRC 服务器发布控制命令操纵 Bot 计算机,利用 Bot 计算机完成很多操作,比如攻击传播 Bot 程序。利用 Bot 进行时段出租,散发垃圾邮件,或者进行 DDoS 攻击等。目前发现的机器人网络(Botnet)规模最大可达数十万台机器。

机器人网络发动 DDoS 攻击示意图如图 5-6 所示。

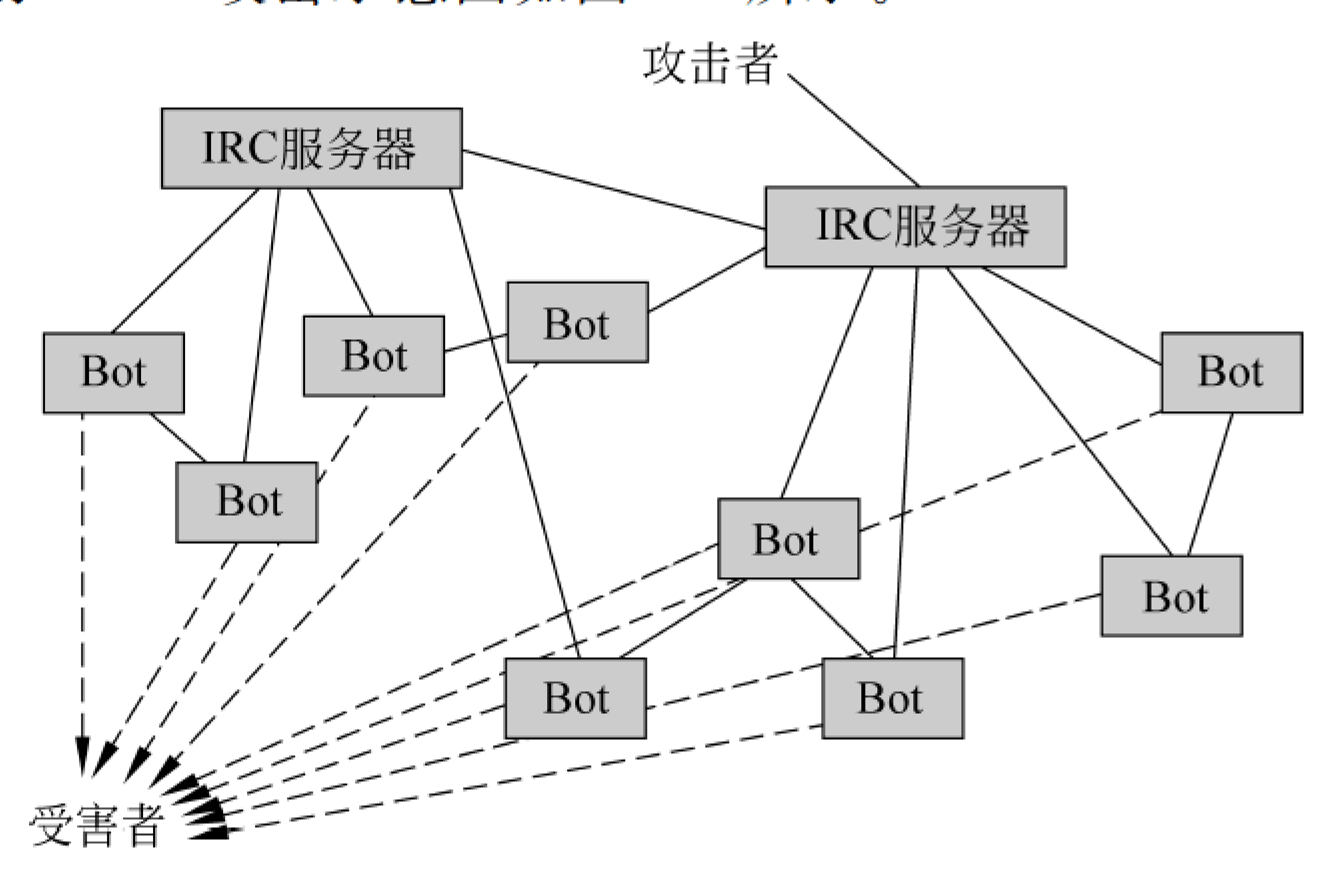


图 5-6　机器人网络发动 DDoS 攻击示意图

5.5 拒绝服务攻击

DoS 是拒绝服务(Denial of Service)的简称,造成 DoS 的攻击行为被称为 DoS 攻击,其目的是使计算机或网络无法提供正常的服务。

常见的 DoS 攻击如下。

1. 网络带宽攻击

网络带宽攻击指以极大的通信量冲击网络,使得可用网络资源都被消耗殆尽,最后导致合法用户无法访问网络资源。

2. 连通性攻击

连通性攻击指用大量连接请求冲击服务器,使得所有可用的操作系统资源都被消耗殆尽,最终使服务器无法再处理合法用户请求。

5.6 分布式拒绝服务攻击

分布式拒绝服务(Distributed Denial of Service,DDoS)攻击指借助于客户/服务器技术,将多个计算机联合起来作为攻击平台,对一个或多个目标发动 DoS 攻击,从而成倍地提高拒绝服务攻击的威力。

(1) 攻击者使用一个偷窃账号将 DDoS 主控程序安装在一个计算机上。

(2) 攻击者秘密安装代理程序到 Internet 的大量计算机上,代理程序能都对目标机器发动 DoS 攻击。

(3) 利用客户/服务器技术,主控程序几乎同时激活运行成百上千的主机上的代理程序,控制代理程序对一个或多个目标发动 DoS 攻击。

5.7 杀毒软件

杀毒软件主要有 Symantec、Mcafee、NOD32、Kaperskey 等。

杀毒软件最核心的内容是获取恶意代码的特征(Signature),恶意代码样本的收集是产生用户端杀毒软件的基本手段。

杀毒软件公司通过全球部署蜜罐网络来捕捉恶意代码,收集杀毒软件从用户端上传的恶意代码等,进行分析,从而提取特征。杀毒软件公司病毒特征库的制作流程如图 5-7 所示。

在分析生成特征之后,生成病毒更新包,下载应用到用户的杀毒软件的扫描引擎中。所以病毒特征的更新对于杀毒软件的防护能力非常重要。杀毒软件公司病毒特征更新流程如图 5-8 所示。

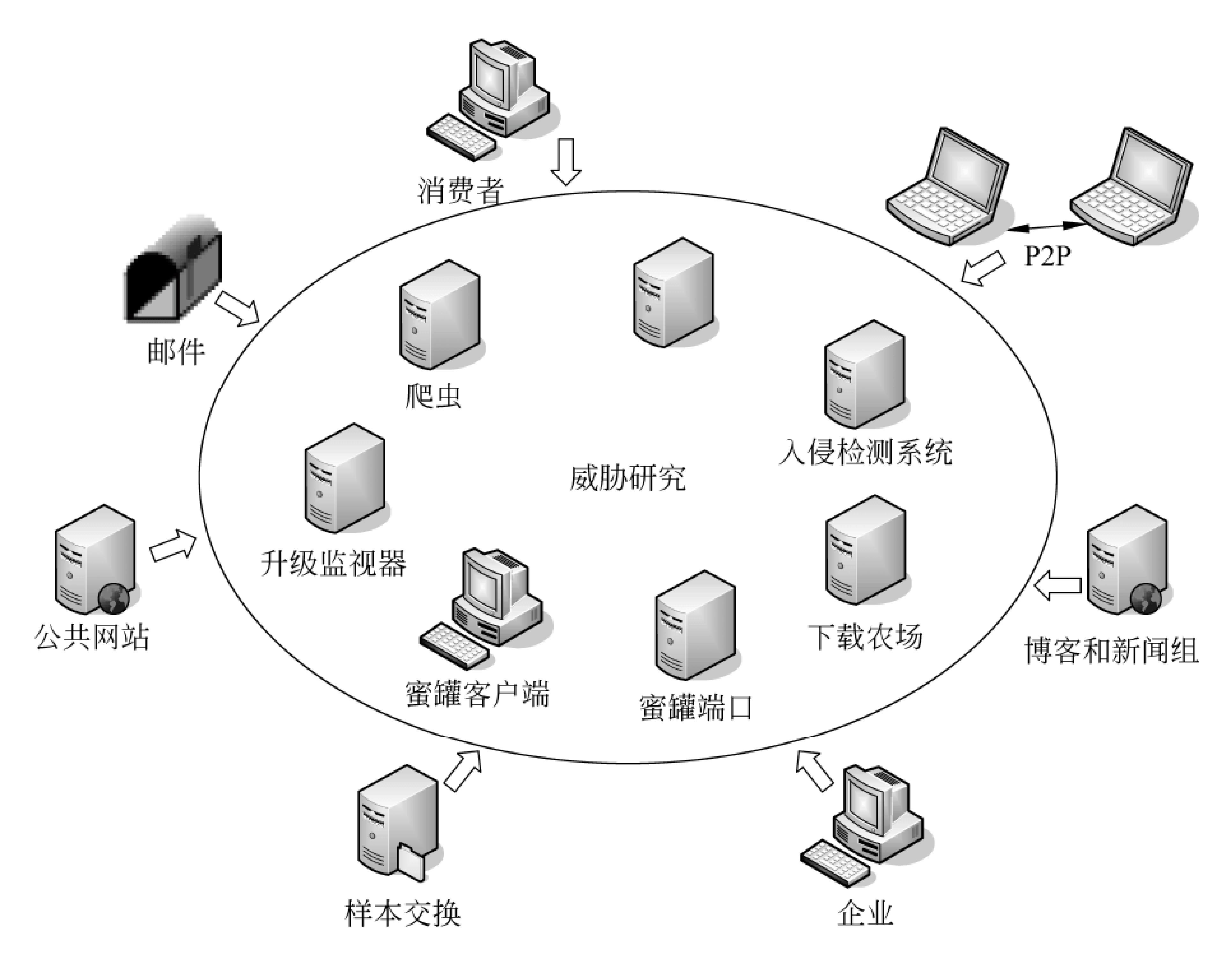


图 5-7　杀毒软件公司病毒特征库制作流程

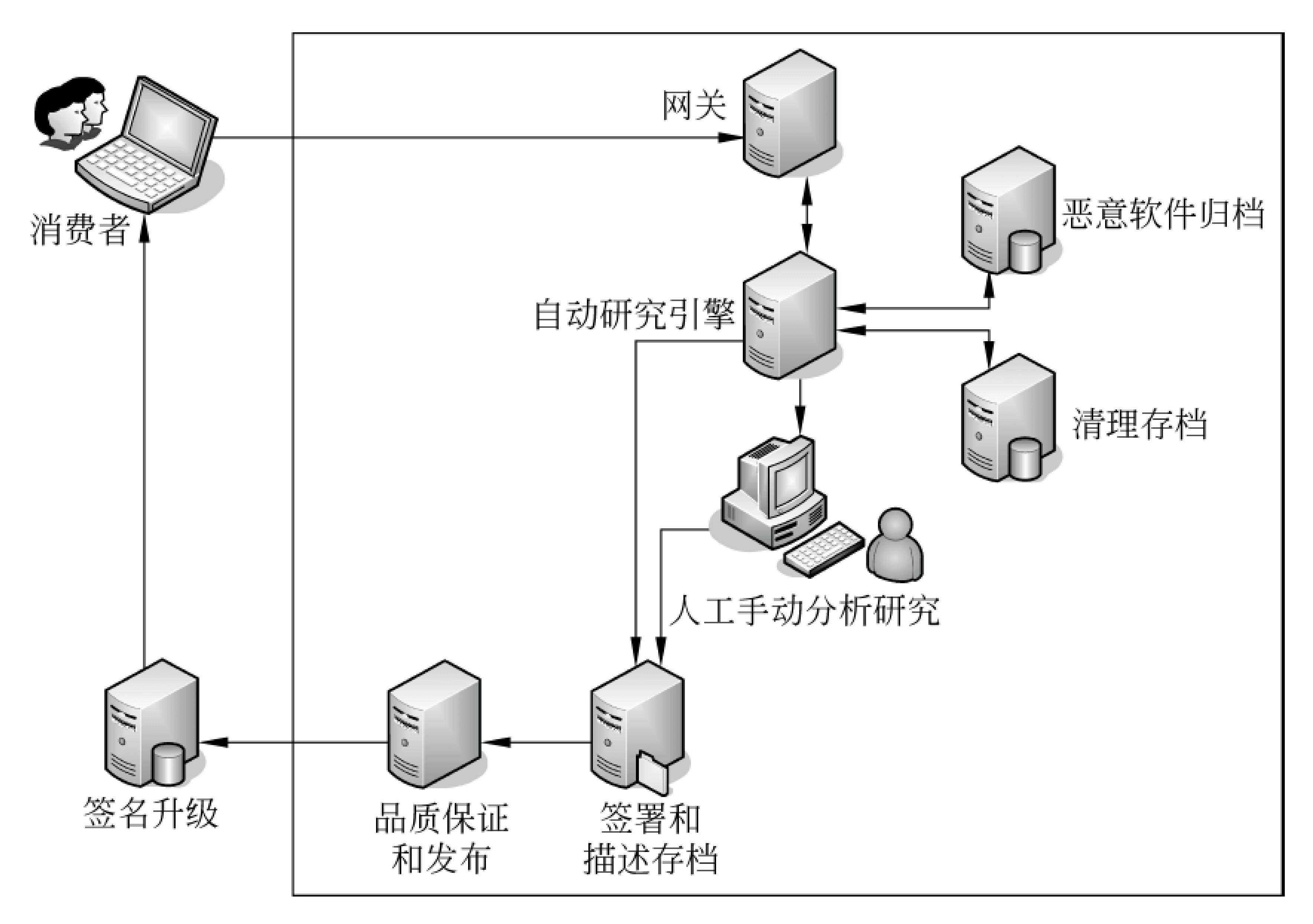


图 5-8　杀毒软件公司病毒特征更新流程

5.8　防　火　墙

通俗地讲，防火墙是介于其他网络和防火墙所保护的网络之间的中界点，控制保护网络和其他网络之间的数据包交换的网络设备。

严格地讲，IETF RFC 2979 定义了防火墙的行为和需求（Behavior and Requirement），

基于以上规范的软件或是硬件，称为防火墙。防火墙能够防止未经允许的连接进入由防火墙所保护的网络，防火墙提供了穿透(Transparency)的功能，并提供了限制穿透的功能。防火墙的基本原理如图 5-9 所示。

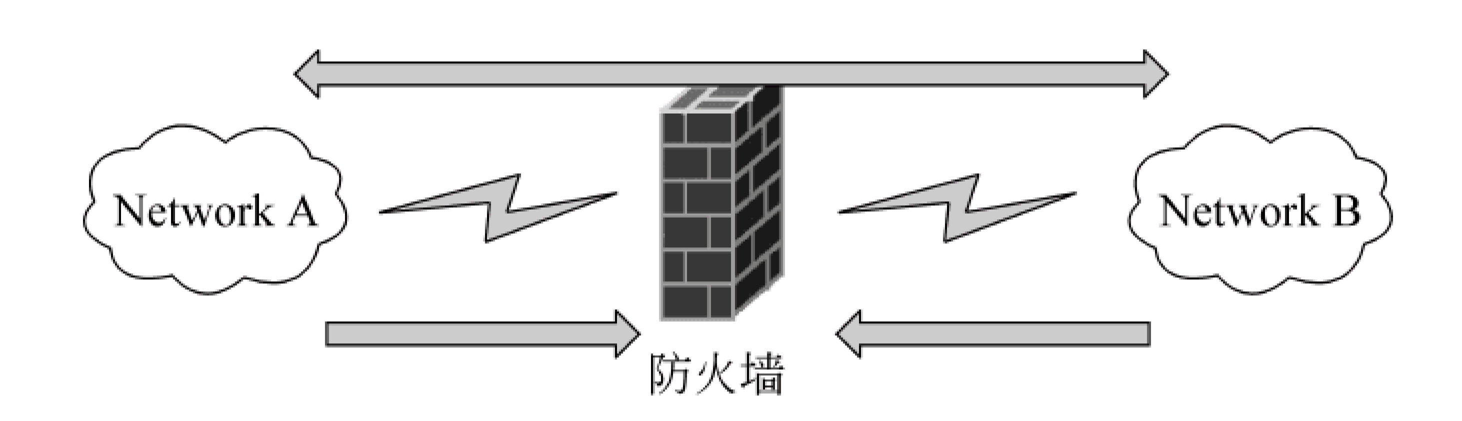


图 5-9　防火墙的基本原理

性能较好的是基于硬件的防火墙，如 NetScreen 系列。

目前个人防火墙也日益普遍，如 Windows 自带防火墙、Norton Internet Security、瑞星个人防火墙等。

5.9　入侵检测系统

1. Snort

Snort 最初由 Martin Roesch 开发，是目前使用最广的开源入侵检测系统项目，规则集比较丰富。很多商业的入侵检测系统也采用 Snort 系统。

2. Bro

Bro 入侵检测系统是加州大学 Berkeley 分校的 Vern Paxson 教授开发的入侵检测系统，是目前学术味比较纯的系统，详细信息访问 www.bro-ids.org。

3. pfense

基于 Free BSD 的流量管理控制软件。

5.10　蜜罐网络

蜜罐(Honeypot)是指伪装的不安全计算机系统，用于吸引攻击，观察攻击的网络工具。蜜罐网络(Honeynet)用多个蜜罐组合起来的网络系统，用来采集攻击者的信息。

蜜罐网络的关键技术有网络虚拟化、端口重定向、报警、数据控制和数据捕获等。

一般实验采用虚拟机(VM)虚拟出蜜罐网络，其中有防火墙、入侵检测系统和多个服务器。

加州大学 Berkeley 分校的 Vern Paxson 教授组建的 GQ 蠕虫分析系统，就是一个类似蜜罐网络分析系统，通过从互联网采集目的地址无应答的流量(疑似蠕虫流量)，将该流量重放(Replay)到蜜罐网络系统，观察其流量引发的连锁反应，以确定是否为蠕虫流量。GQ 蠕

虫分析系统如图 5-10 所示。

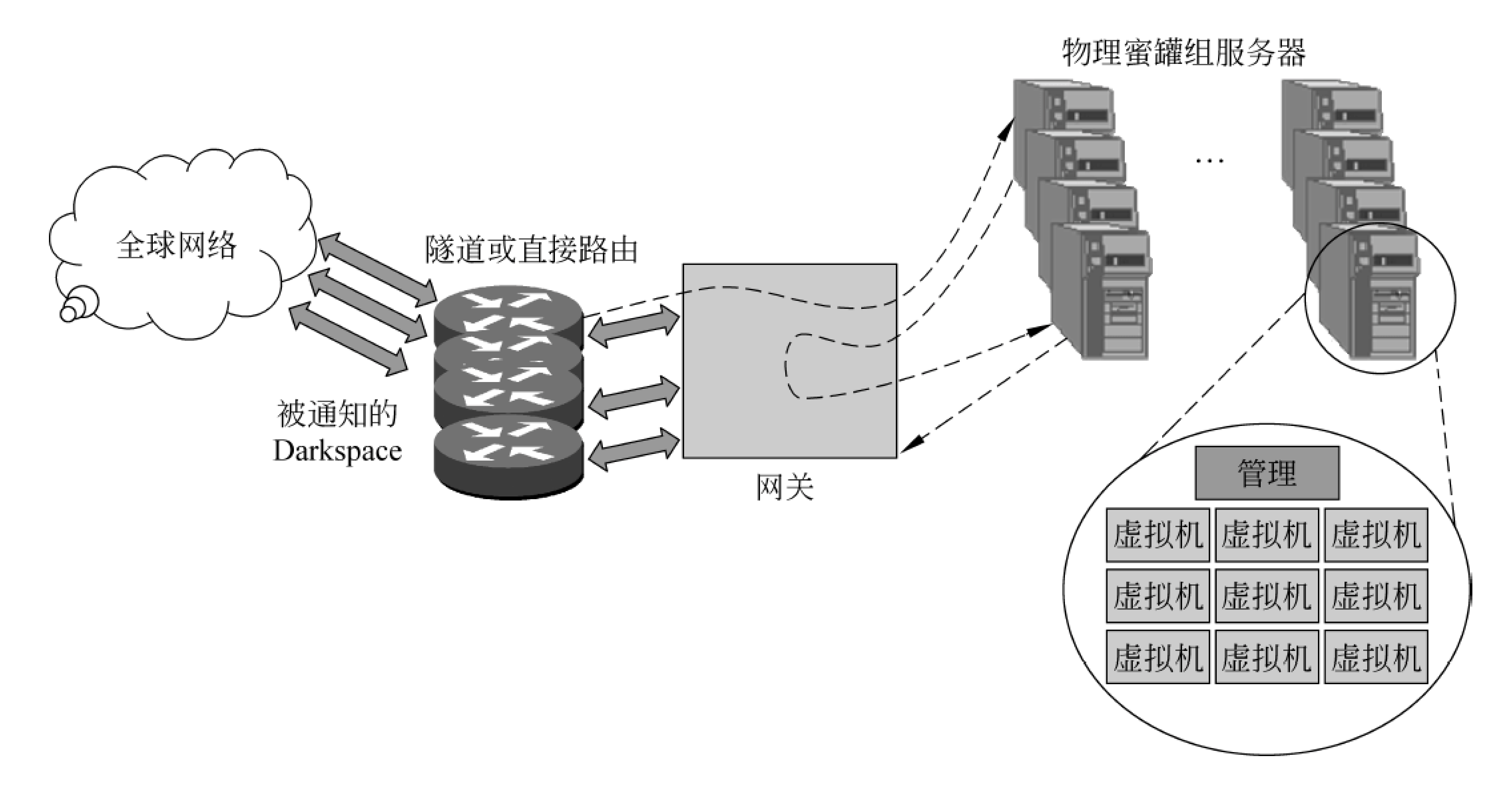


图 5-10　GQ 蠕虫分析系统

其他蜜罐网络,如北京大学的狩猎女神项目也很受关注。

第6章　互联网安全影视

6.1　黑　　客

“黑客”一词是由英语 Hacker 音译而来，原意是指用斧头砍柴的工人，最早被引进计算机圈则可追溯自20世纪60年代。最初的黑客一般都是一些高级的技术人员，他们热衷于挑战、崇尚自由并主张信息的共享，他们精通各种编程语言和各类操作系统，伴随着计算机和网络的发展而成长。黑客所做的不是恶意破坏，在黑客圈中，Hacker 一词是褒义的。但在媒体报道中，黑客一词往往指那些“软件骇客”（Software Cracker）。到了今天，黑客一词已被用于泛指那些专门利用计算机网络搞破坏或恶作剧的人。对这些人的正确英文叫法是 Cracker，有人翻译成“骇客”，很多人将黑客与骇客化为一类，实际为错误区分。“黑客”与“骇客”是分属两个不同世界的族群，基本差异在于，黑客是有建设性的，而骇客则专门搞破坏。

《黑客》（1995年）电影描述的是正义者对抗邪恶者的故事，是很早的关于黑客的电影。

电影中，Dade 是一名计算机怪杰，他在11岁那年，盗取了 FBI 的机密档案，被政府当局下令禁止再接触计算机。而另一边的 Kate，也是一名上网高手，通过计算机二人不时展开对垒，比试高低。他们因出色的技术而被卷进一家国际夺网阴谋中。受聘于大集团的首脑 Eugene，利用工作关系上网，并向大财团要挟金钱，否则便破坏其计算机程序，使六艘大油轮爆炸。就在千钧一发之际，Dade 与 Kate 联手夺网成功，避免了一场大灾难。

6.2　防　火　墙

防火墙（Firewall）指的是一个由软件和硬件设备组合而成、在内部网和外部网之间、专用网与公共网之间的界面上构造的保护屏障。防火墙是一种获取安全性方法的形象说法，它是一种计算机硬件和软件的结合，使 Internet 与 Intranet 之间建立起一个安全网关，从而保护内部网免受非法用户的侵入。计算机流入流出的所有网络通信和数据包均要经过防火墙。防火墙实际上是一种隔离技术。它允许你“同意”的人和数据进入人们的网络，同时将人们“不同意”的人和数据拒之门外，最大限度地阻止网络中的黑客来访问人们的网络。

最危险的地方就是最安全的地方。反之则有：最安全的地方就是最危险的地方。类似地有：最安全的人就是最危险的人。所有安全因素中，最脆弱的因素就是人，当人作为安全规则的一环被突破时，防火墙也就变成了放火墙。

《防火墙》（2006年）由哈里森·福特主演，其讲述了一位银行金融信息主管与犯罪分子进行斗争的情节。

年过中旬的杰克·斯坦福（哈里森·福特饰）是位于美国西雅图市太平洋银行的网络安全高级主管，他凭借其主持设计的高科技防火墙软件而备受高层的重视，也因此杰克与妻子

儿女过着养尊处优的幸福生活。然而，他的重要性也同样引起了不法之徒的注意。歹徒比尔·考克斯是个高智商的罪犯，他和他的团伙用一年的时间研究斯坦福一家的作息习惯和相关资料。在一切准备充分后，比尔绑架了杰克的家人，并胁迫杰克破解太平洋银行的防盗系统，从而盗出一亿美元现金。顾及妻儿的性命，杰克违心答应了比尔的要求，与此同时他也和这个狡猾的家伙展开了斗智斗勇的对决。

6.3 黑客帝国

《黑客帝国》(1999 年)由基努·里维斯主演，本质上讲的是新型智能生命进化的过程。

不远的未来，智能机器和人类爆发了大战，机器的智能使得它们利用现有科学技术的能力极其强大，从而将人类击败。但是它们的创新能力严重匮乏，无法提出更高级的理论。

机器智能的思维严格遵循逻辑运算，对无用的程序，一律删除。这种模式严重阻碍了理论创新能力的发展。但放任无目的程序存在和泛滥，可能会导致无法预料的毁灭性结果。机器智能文明处于两难境地，严格执行无用程序删除的原则，会令文明永远只能在一个水平上重复，而不会出现革命性的进步。而不严格执行该原则，则整个机器文明的生存都有可能受到威胁。

出于以上原因，机器文明设立矩阵系统尝试解决这个问题。利用人类在矩阵系统中产生的创新思维，为机器的进一步发展提供思想动力。机器文明认为如果系统运行顺利，就可以一直利用人类的思维，否则就采取矩阵升级的战略，通过矩阵革命，产生一个能够自主创新的矩阵系统，从而彻底摆脱对人类的依靠。在这个大背景下，主人公尼奥的选择将决定人类和机器的未来。

该片中有大量的创意点可以和现有的计算机与网络技术相对应，通过该片可以大大增进互联网安全课程的趣味性。

6.4 操作系统革命

《操作系统革命》(2001 年)是一部由 J. T. S·摩尔(J. T. S. Moore)导演的纪录片，该片追述了 GNU、Linux、自由软件运动以及开放源代码运动长达 20 余年的历史。在微软公司垄断下有一件东西永远不会给你真正的自由。也正是因为这个原因，不少先锋人物站出来反抗微软帝国，并努力建立一种新的操作系统——没有人为的限制，任何人都可以自由地使用。《操作系统革命》向公众介绍这些建立 Linux 操作系统，奋起反抗垄断的斗士的人生经历。现在微软公司已经明显感到了来自 Linux 的压力。微软公司的首席执行官曾公开表示："Linux 是一种癌症！"但这丝毫不能影响 Linux 发展的步伐。《操作系统革命》访谈了 Richard Stallman 这位自由软件基金会的创始人、自由软件运动的发起者，GNU Emacs 等软件的作者；开源运动的先驱 Eric Raymond，《大教堂与集市》(Cathedral and Bazaar) 的作者；Linus Torvalds，这位 Linux 内核的最初作者，以及 Open Source 开源项目的由来和 Linux OS 的商业运作等。

有问有答

下面就人们感兴趣的问题进行解答，以提高交互性。

1. 如何保证个人计算机的健康性？

千万注意从网络上随便下载软件的风险。因为这些软件的可信性不能验证，很可能其中被植入了木马或者病毒。原来流行很广的一键安装的软件版本都曾爆出其中存在后门。

建议采用正版软件。从比较可靠的软件源去下载软件，如清华大学电子系 FTP、清华大学自动化系 FTP 等。

在当前的网络环境下，杀毒软件、个人防火墙等软件工具不可缺少，病毒库需经常更新；关键的系统补丁和应用程序补丁也要经常更新。这是当前互联网安全防御的无奈之举。

另外，也可以选择开源免费的 Linux 操作系统，但需要相关的知识。推荐使用 Ubuntu 软件，只要在网上注册，就能收到最新的 Ubuntu Linux 安装光盘。校内也有不少 Linux 软件源可以用。

2. 如何建立网络空间的信任？

信任(Trust)是整个社会正常运行的基本社会关系，按照一般定义：一个事物是可信的，意味着其身份(Identity)是可确定的，其行为(Behavior)是可预期的。

信任在电子世界网络情景下依然适用，在网络空间中信任关系的载体，信任关系的建立，确认，维护，传递和终止等过程，是通过公钥基础框架(Public Key Infrastructure, PKI)和数字证书等技术来实现的。

PKI 采用类似公安局颁发公民身份证的方法，在网络中的对应物是数字证书(Digital Certificate)。简单地说，数字证书是权威机构 CA 颁发给个体用户的，个体用户之间用来辨识身份的证明。在网络上通过交互，验证数字证书，可以在一定程度上保证交互双方的可信性。

典型的一份 X.509 数字证书 THUCA(在 Info 门户中使用)可以用如下方法打开。打开 IE 9 浏览器，进行如下操作：

单击“工具”→“Internet 选项”→“内容”→“数字证书”。

颁发数字证书的权威机构 CA 是 PKI 框架的核心，权威机构 CA 之间的一种等级结构就是 PKI 集中式信任体系模型，如图 6-1 所示。

3. 可信计算能够创建可信的网络空间吗？

可信计算(Trusted Computing)通过 TPM 技术，能够增强计算平台的可信性。

(1) 可信计算平台 TP 应该采用内嵌的硬件化的密码模块确保身份与根信任，类似于手机终端的 SIM 卡，能够唯一标识每个计算平台。基于硬件密码模块的方案比软件方案更安全。

(2) 可信计算平台 TP 从系统启动开始进行硬件与软件的验证。

(3) 可信计算平台通过平台身份认证来向交互方证明自己的可信性。

采用 PKI 框架，采用信任状或证书(Credential & Certificate)来保证交互双方的身份和平台的安全性。

但是，可信计算也存在着很多问题。

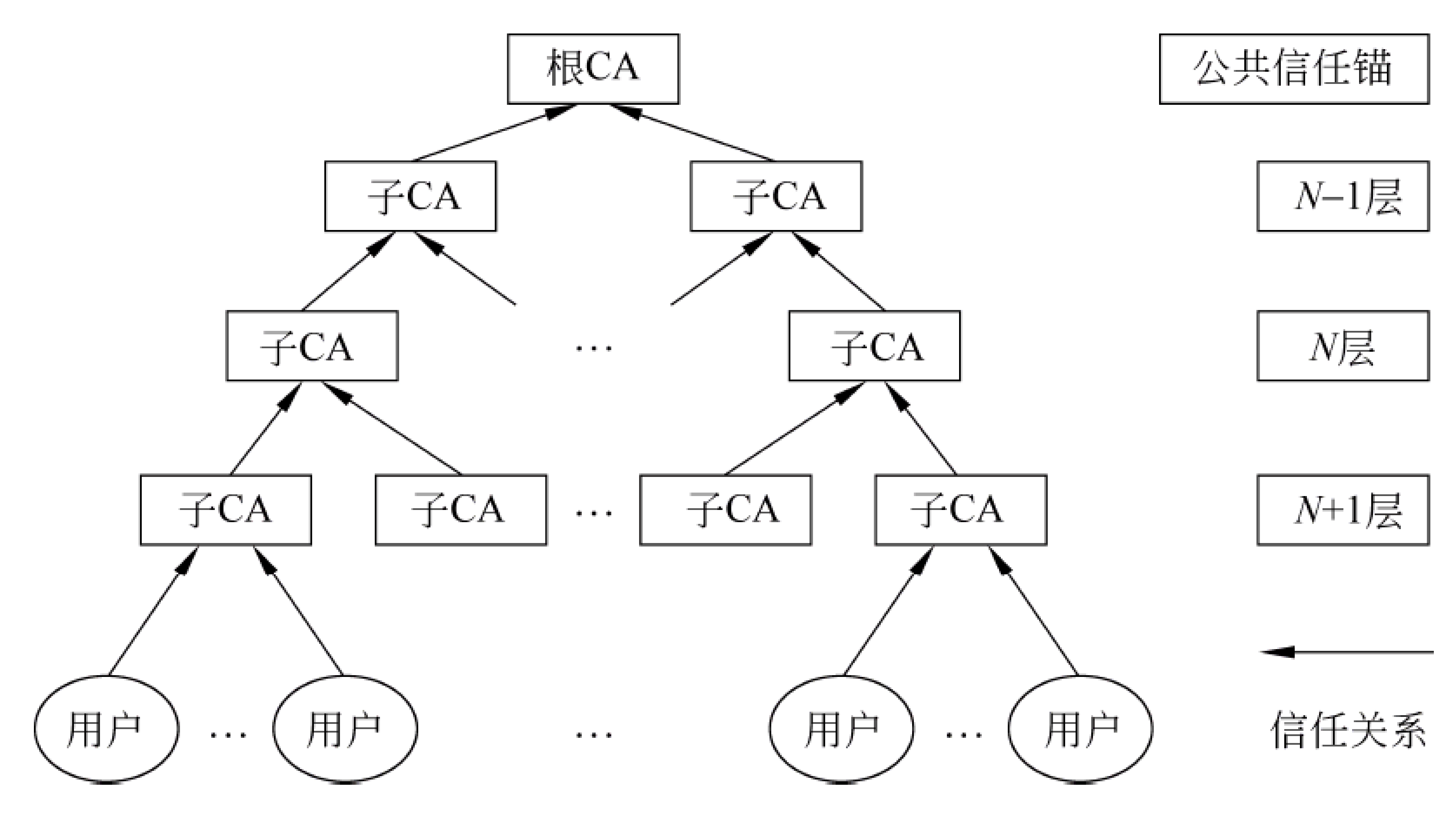


图 6-1　PKI 框架可信性认证体系

常用可信计算应用有 Microsoft 的 BitLocker 文件加密系统。目前绝大多数笔记本都安装了 TPM 芯片，如果要使用只要在 BIOS 中启用 Security Chip 就可以了。

4. 网络银行安全吗？

网络银行存在着安全风险。当前网络银行有几种手段来保证安全，例如，数字证书、U 盾、USB Key、动态口令、手机提醒等手段。目前国内网络银行的用户界面多采用在 IE 浏览器中嵌入 ActiveX 控件或者采用客户端软件的方式，因此在键盘输入和客户端软件之间可能挂上钩子（Hook），导致密码被盗，即使采用鼠标键盘的方式，也不可能完全避免。此外，IE 浏览器之间的窗口存在着安全隐患，运行在其中一个 IE 窗口的恶意软件有可能获取其他 IE 窗口的内容信息。

可信计算采用平台验证的方式，保证平台上运行的软件不被非法篡改，且将密钥存放在硬件中（同 U 盾的方法），一定程度上提高了攻击者的攻击门槛。

最简单的保证平台健康性的方法是进行网银的业务操作的机器尽量不要用来玩游戏或者他用，在一定程度上可以保障安全。

5. 802.1x 端口认证是什么？

802.1x 就是 IEEE 为了解决基于端口的接入控制而定义的一个标准。802.1x 首先是一个认证协议，即一种对用户进行认证的方法和策略。它是对端口进行控制的。这里的端口可以是实际的物理端口也可以是虚拟的 VLAN 端口。“基于端口的网络接入控制”是指在局域网接入设备的端口这一级对所接入的设备进行认证和控制。连接在端口上的用户设备如果能通过认证，就可以访问局域网中的资源；如果不能通过认证，则无法访问局域网中的资源。

802.1x 体系结构如图 6-2 所示，它包括三个实体：客户端、设备端和认证服务器。客户端一般是指网络终端，是需要接入网络的设备，客户端需要支持 EAPOL（Extensible Authentication Protocol over LAN，局域网上的可扩展认证协议）协议，需要运行 802.1x 客户端软件，图中 PAE（Port Access Entity，端口访问实体）是认证机制中负责执行算法和协议操作的实体。设备端通常为支持 802.1x 协议的网络设备（如 H3C 系列交换机、思科 Aironet 系列无线接入点），它为客户端提供接入局域网的端口。认证服务器是为设备端提供认证服务的实体，用于实现用户的认证、授权和计费，通常为 RADIUS 服务器。

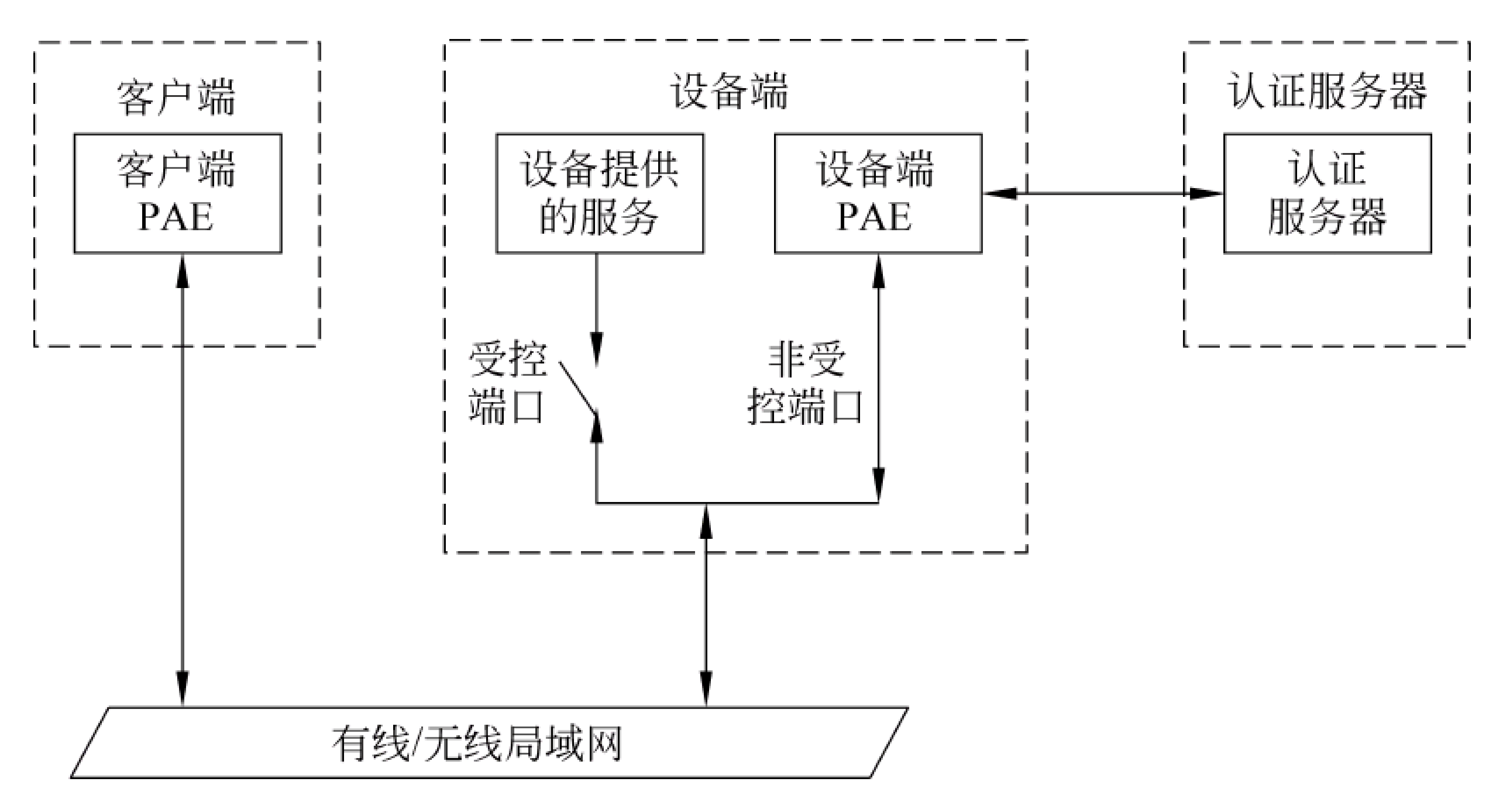


图 6-2　基于 802.1x 的网络接入原理

6. 在网络上人们之间的通信安全吗？

答案是没有绝对安全。如何提高通信安全，可以采用以下方法：如私有的通信协议（增加分析成本），构架私有的服务器（提高追踪的难度），对通信进行高强度的加密和扰动（产生很多无用的数据和强加密数据）。

7. 能对所有的通信进行监控吗？

理论上可行，实际中不可行。这也是导致当前互联网不能立即升级为更高带宽的原因。

监管有正反两面：能够有效减少网络的攻击，但可能会对公民的隐私权有所保留。另外，监管权利的滥用也是一个问题。

展　　望

互联网安全原理探究课自 2006 年设立以来，经过 7 年多的反复教学（近 100 堂课），已经为 2000 多名本科生开设了讲座，选课的同学来自经管、法律、新闻、社会、计算机等各个专业。同学们在提交的报告中纷纷指出该课程信息量太大，需要提交相应的导读手册；同时也纷纷指出该课收益很多，对切身的互联网安全问题有更深的了解。分析同学们提出的问题主要原因是互联网安全牵涉的基础知识太多，因此引文内容，铺垫部分需要花很大精力去讲，否则后面的内容讲不清，同学听不明白。因此特整理这本导读手册，供大家参考。

实验室探究课作为国家精品课程，其主旨是将专业知识大众化，将科研工作的成果以教学探究的方式与同学研讨，一方面可以总结整理科研工作者的研究工作，发现自身的不足和可能的研究方向；另一方面，丰富了学生的知识，开阔了学生的视野，并通过“探究课”的指导，增强了学生的实际分析和研究的能力。

我们坚持计算机工程科研方面，以系统工作为中心，以实际系统为研究对象，在 QoSLAB 与 NSLAB 先后完成了反蠕虫系统 AntiWorm@THU、安全配置核查 SecuConf@THU、可信网络访问 TNC-NAC@THU、网络探针 SNIFFER@THU、统一威胁管理 UTM@THU 等系统原型，项目经验教训的积累为本课程的写作提供了极其丰富的素材。

因为时间有限，再加上信息技术日新月异，疏漏或不妥之处还请大家指正。

中　　篇

互联网安全平台配置

第 7 章　UTM 简介

Untangle UTM 一体化安全网关提供单机管理和远程管理相结合的双重管理机制。主要安全功能包括入侵保护、防病毒、防火墙、协议控制、服务质量、虚拟专用网、流量记录、攻击拦截、网页过滤、广告拦截、防间谍软件、垃圾邮件拦截、钓鱼网站拦截和日志图表报告。

7.1　UTM 介绍

目前主要的网络安全问题有黑客攻击(Hacking)、恶意代码(Malware)(包括特洛伊木马(Trojans)、蠕虫病毒(Worms)、间谍软件(Spyware)、流氓软件(又叫做 Adware)、网络欺诈(Phishing Attack)和网络攻击(Attack)(包括 DDoS、僵尸网络 Botnet)等。

针对这些问题,网络安全防护提出了相应的解决方案。安全威胁日益复杂多样,攻击手段日益综合。很多复合式安全攻击融合了如蠕虫病毒、木马、间谍软件等多种手段,单纯依靠以单一的防火墙为代表的传统安全解决方案已经无法奏效。而采购、部署和管理多种单独防护手段(如反钓鱼、反蠕虫、IPS 等)安全设备往往需要企业花费巨大的经济和人力开销,并需要专业的网络管理知识,从而给企业带来了巨大的维护成本。

针对上述问题,2004 年 9 月 IDC 提出了一体化的安全设备——UTM。UTM 是统一威胁管理(United Treatment Management)的英文缩写,UTM 是一种多功能安全网关设备,包括防火墙、VPN、网关防病毒、IPS、访问控制、内网监控等多种安全功能。UTM 的基本原理与功能如图 7-1 所示。

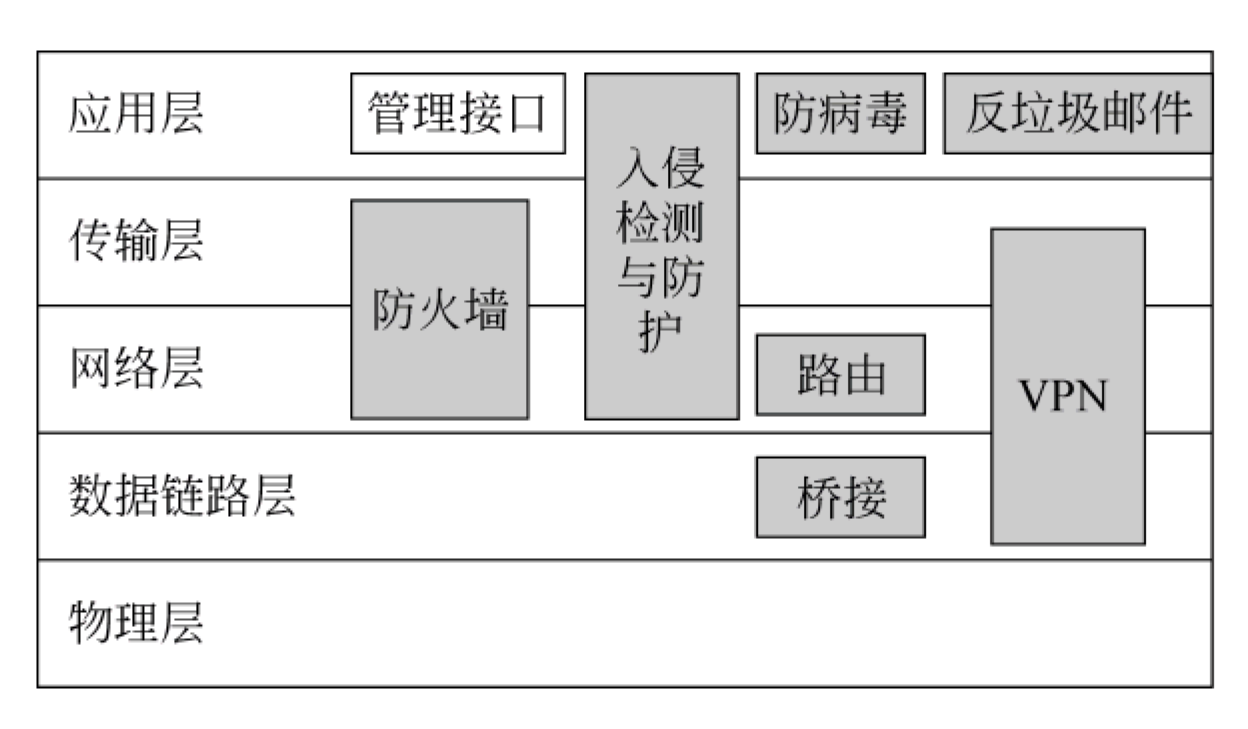


图 7-1　UTM 的基本原理与功能

7.2　默认部署图

在高速互联网入口和接入网络环境中,为了阻挡来自互联网的安全威胁以及对各种业务应用、Web 2.0、VoIP 等应用,需要全面的安全策略,包括多层网络防御。同时,为了保障许多新型应用的有效性和安全性,迫使管理员部署高效多层防御的安全设备。UTM 系列

网络安全解决方案提供实时的多层安全威胁防御机制。Untangle UTM 默认的部署模式如图 7-2 所示。

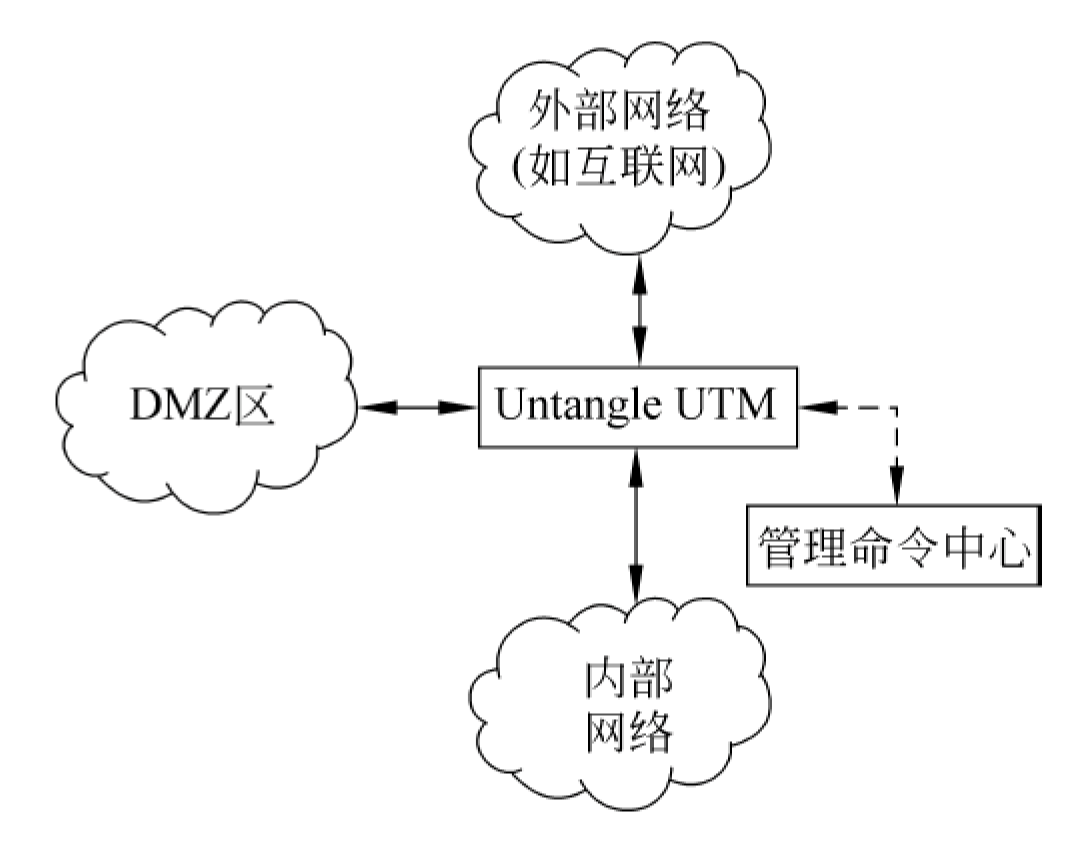


图 7-2　Untangle UTM 默认的部署模式

Untangle UTM 依据对现有网络的影响程度,在网络中部署有三种模式,可以采用路由器(Router)、网桥(Bridge)和重路由(Re-router)模式,如图 7-3 所示。

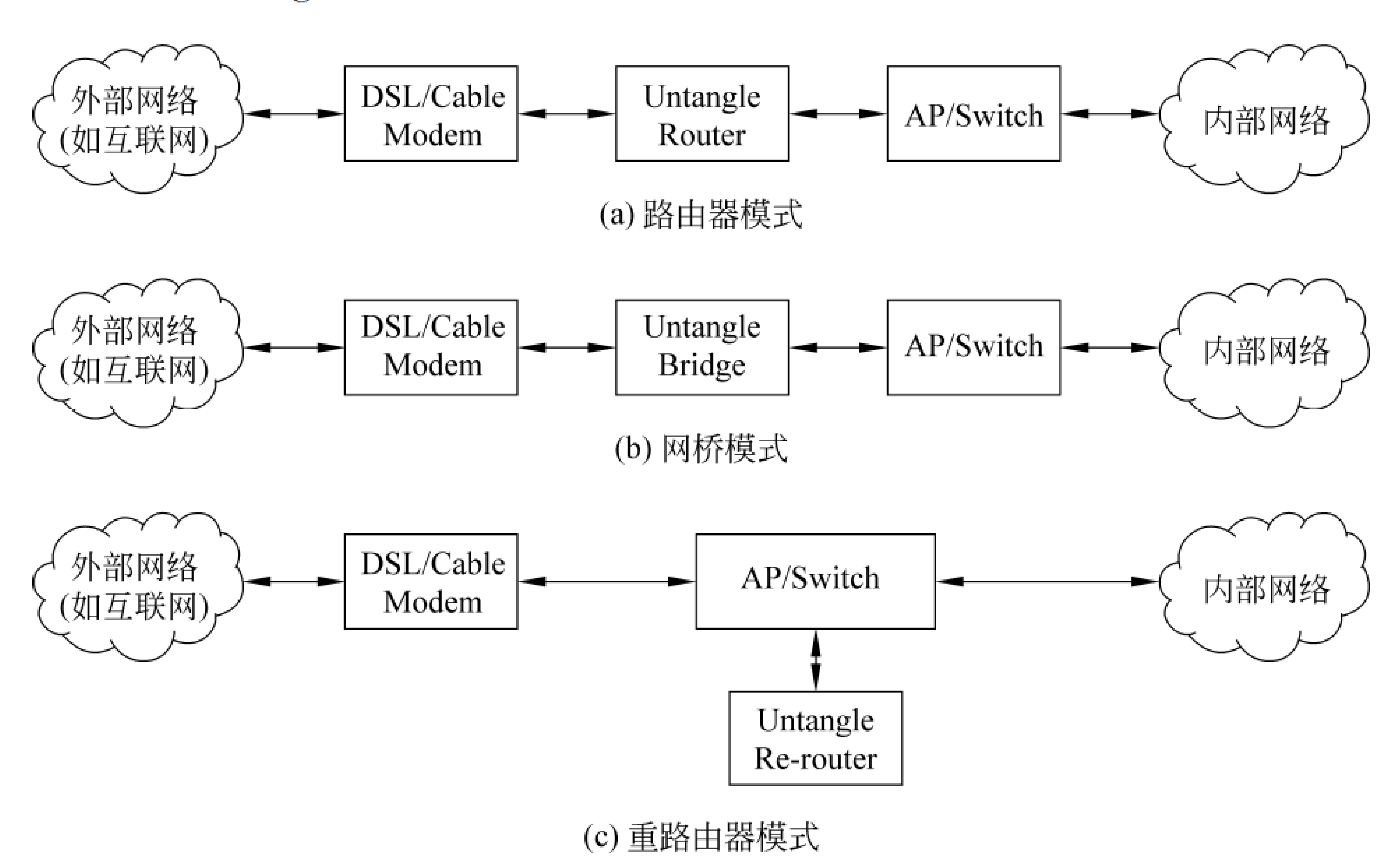


图 7-3　Untangle UTM 的三种部署模式

第 8 章　UTM 的配置

8.1　串口模式管理界面

使用串口线连接 UTM 的前端面板串口(Console),查看默认 IP 地址设置。

使用串口线连接 UTM 的前端面板串口,并将串口波特率设为 9600,可连接设置查看默认 IP 地址,可见串口超级终端界面,如图 8-1 所示。

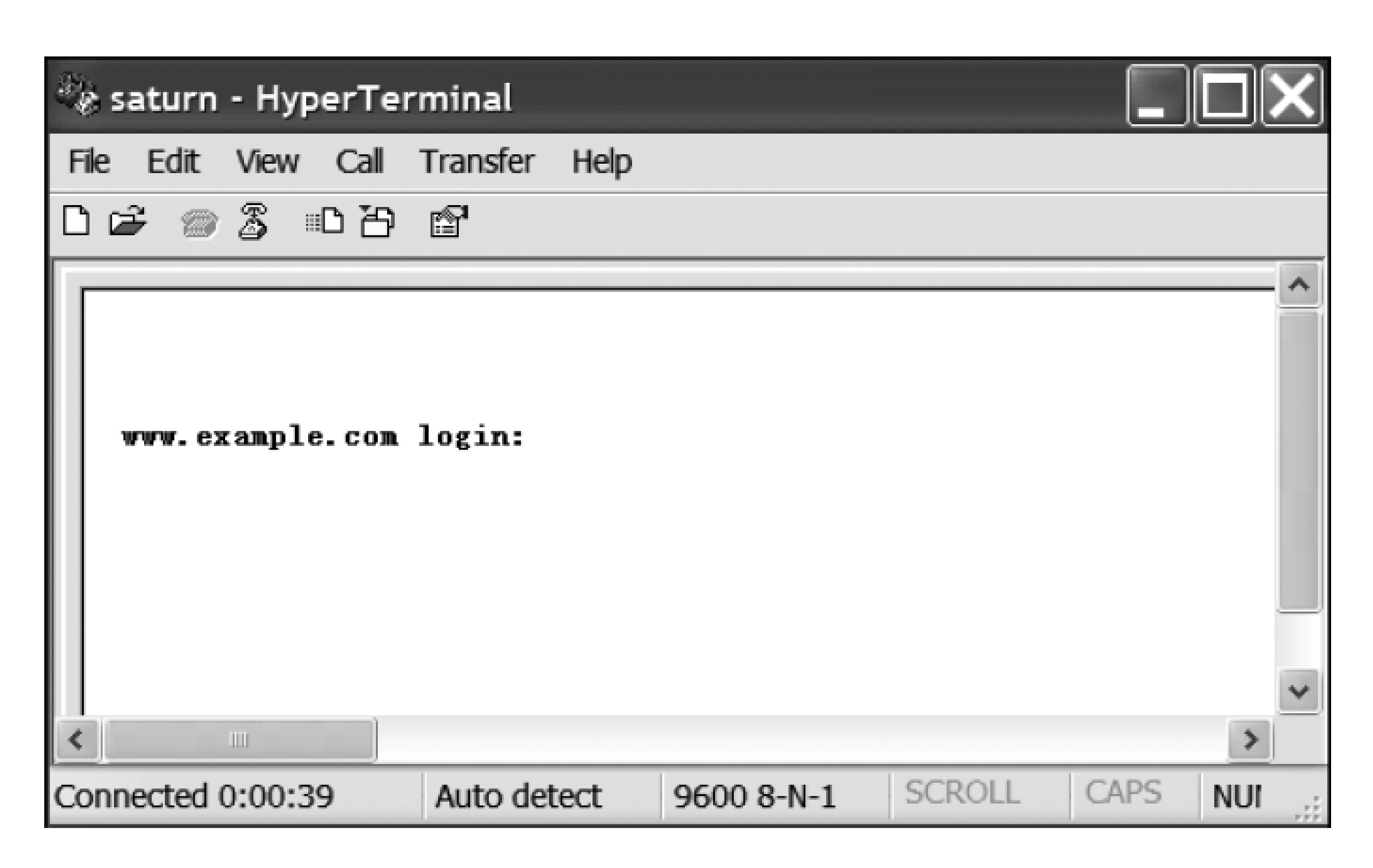
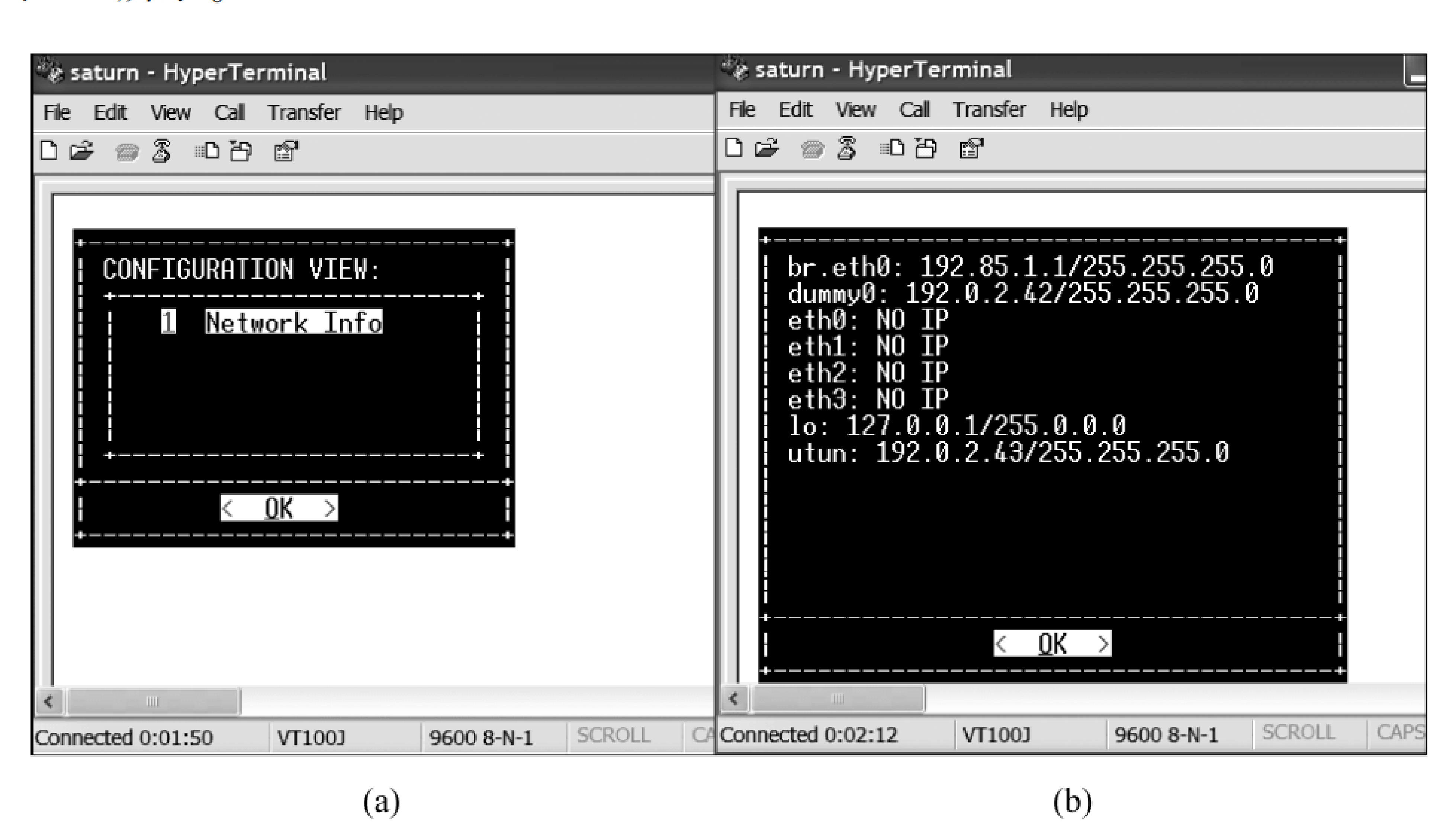


图 8-1　串口超级终端界面

设备启动后初次连接到串口提示需要输入用户名。

输入任意用户名后按 Enter 键进入选项菜单,可查看各网口 IP 配置。选项菜单与查看状态如图 8-2 所示。



(a)　　(b)

图 8-2　网口默认 IP 地址配置状况

8.2 登录 Untangle UTM

8.2.1 登录界面

将 UTM 的配置口与管理机器直接相连(采用交叉网线),或者将 UTM 和管理机器同时连接到一个交换机上,配置管理机器为 192.168.2.*,用浏览器访问 http://192.168.2.1,登录界面如图 8-3 所示。

输入默认管理员用户名 admin、密码 admin,进入 UTM 主界面(为了安全起见,建议第一次登录后,立即修改密码)。

主管控界面分为配置板块(左边)与功能板块(右边)两大块。配置板块用于对 UTM 设备进行网络相关的设置与配置。功能板块展示了已安装的功能模块和服务。



图 8-3 Untangle UTM 登录界面

8.2.2 主界面

启动成功后,将看到如图 8-4 所示的主管理界面。单击启动浏览器客户端,出现图 8-3 所示的登录界面。

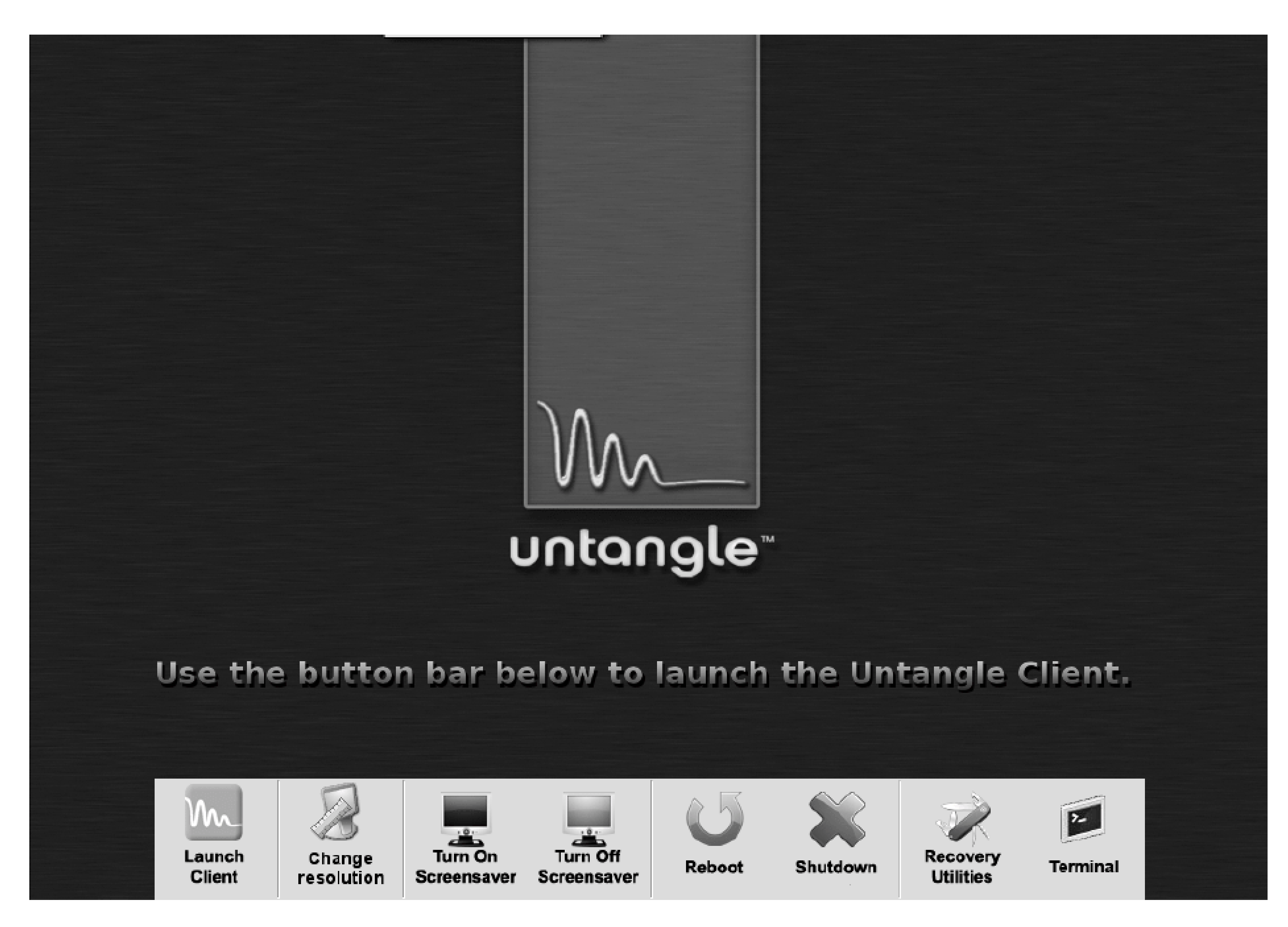


图 8-4 Untangle UTM 主管理界面

输入用户名和密码,成功登录后,显示如图 8-5 所示的安全功能管理界面。

管理界面有不同风格可以选择,图 8-6 展示了部分模块功能(不同视图风格),其主界面

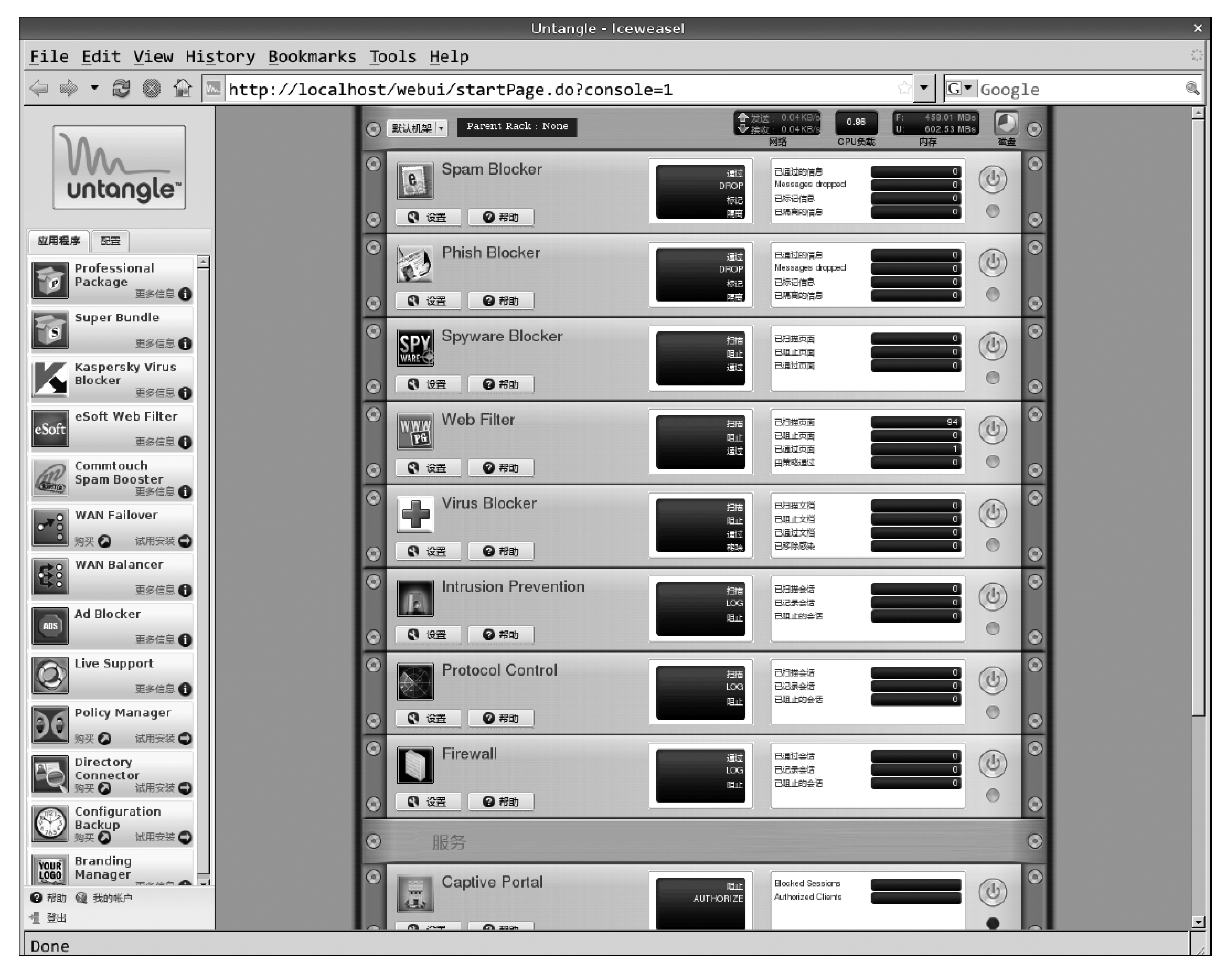


图 8-5　Untangle UTM 安全功能管理界面

如图 8-7 所示。

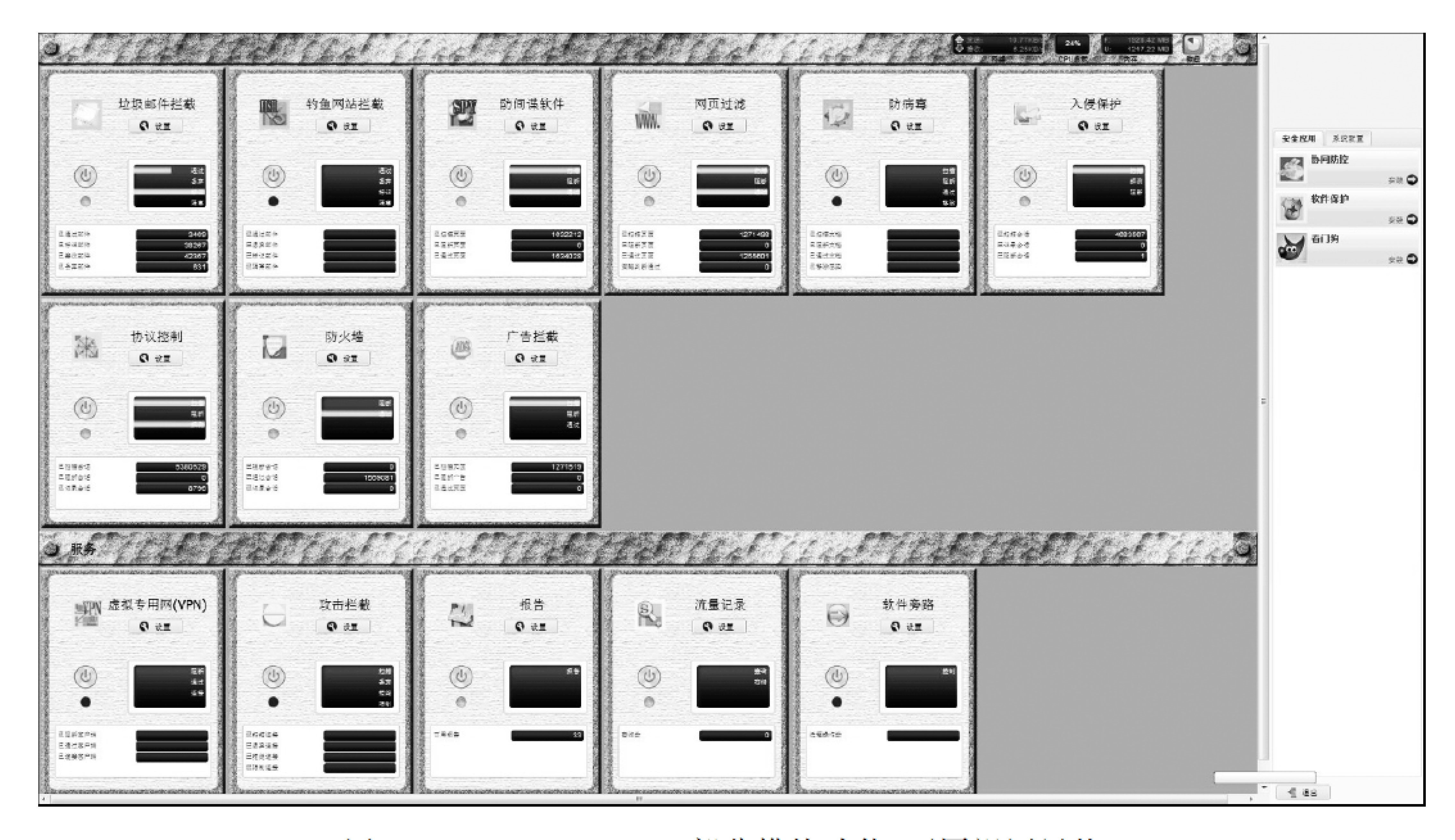


图 8-6　Untangle UTM 部分模块功能(不同视图风格)

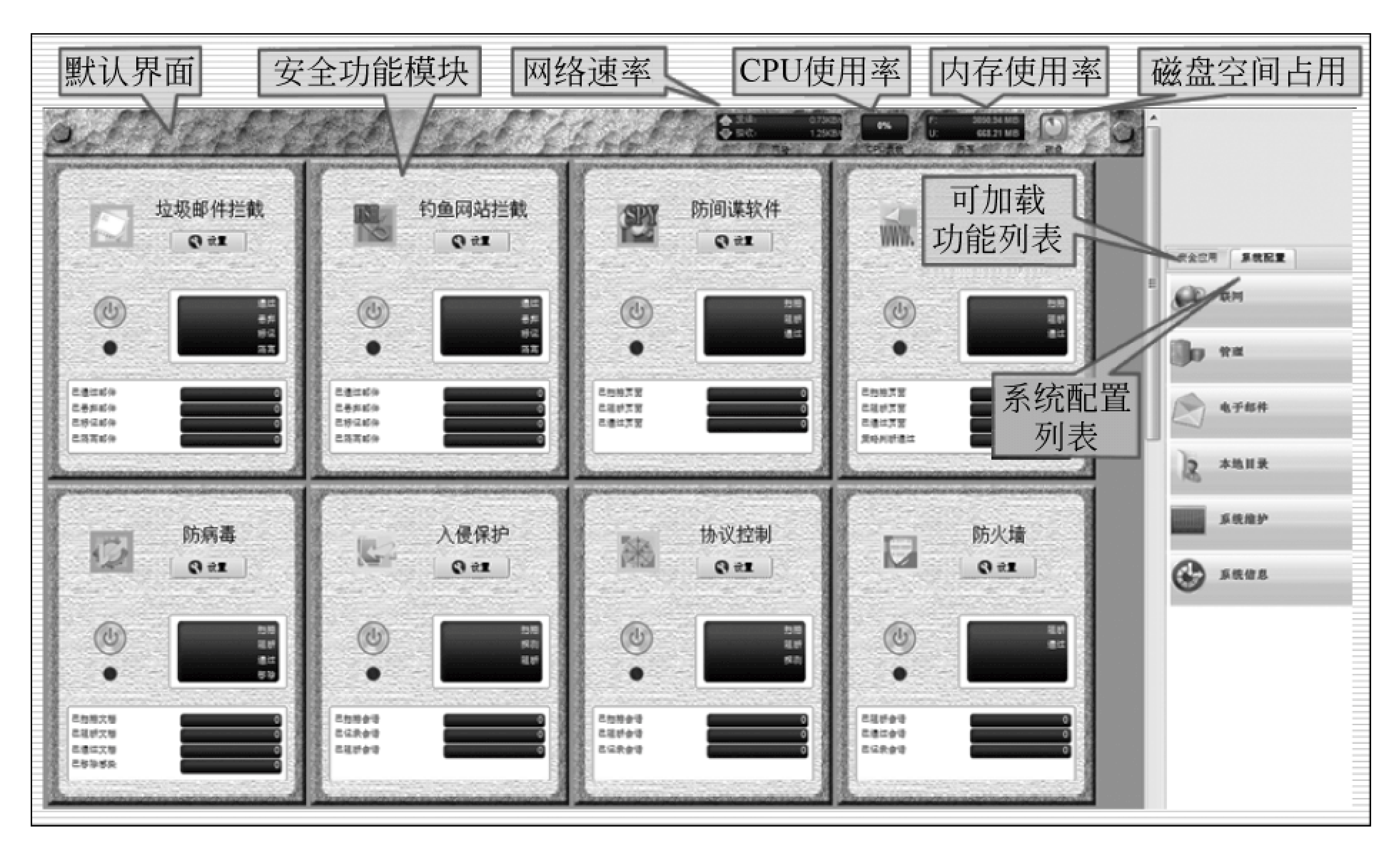


图 8-7　主界面

8.3　系统配置

8.3.1　配置板块菜单

单击右侧"配置"选项卡，出现如图 8-8 所示的配置项。其中，联网和管理最为重要。



图 8-8　Untangle UTM 配置选项

8.3.2　联网配置

联网配置用于配置设备的网络属性。单击图 8-8 中的"联网"，将进入联网配置页面(默认进入网络接口配置界面)，如图 8-9 所示。主要配置项有网络接口、端口转发、主机名、DHCP 服务器、DNS 服务器、QoS 和高级等。其中前 6 项属于标准配置。

网络接口主要可以设置为三种类型。

(1) 外部网口：连接外网的接口，通常是 Internet。

(2) 内部网口：连接内网的接口，通常是受 UTM 保护的网络。

(3) DMZ 网口：非军事化区，通常是用来放置必须允许外网用户公开访问的服务器，如 Web 服务器、FTP 服务器等。它是为了解决安装 UTM 等安全设备后外部网络不能访问内部网络服务器的问题，而设立的一个非安全系统与安全系统之间的缓冲。UTM 通过对流的重定向使得外网用户可以访问部署在 DMZ 的 Web 等服务器。就安全信任程度来说，有如下关系：

未知网口＜外部网口＜DMZ 网口＜VPN＜内部网口

1. 网络接口

Untangle UTM 网络接口配置如图 8-9 所示。



图 8-9　Untangle UTM 网络接口配置

网络接口用于配置接口的类型、地址、工作模式等。网络接口主要分为外部网口(External)和内部网口(Internal),可以通过拖曳列表中的每一行,来改变外部网口和内部网口与物理网上间的对应关系。单击各网口前的“编辑”可进入网口配置页面。

(1) 更新网络接口:对物理接口进行扫描,更新正在使用或未使用的接口。

(2) 测试连通性:检查是否能够连入 Internet。

(3) Ping 测试:测试从 UTM 到指定目的地是否能够 Ping 通。

(4) 编辑:单击后可以对相应网络接口进行多项设置,例如,指定其连接模式、设定其别名等。

1) 外部网口配置

单击“编辑”,可进入该网络接口的配置界面,如图 8-10 所示。



图 8-10　Untangle UTM 网口配置

外部网口配置需要选中网口配置页面“是外网接口”复选框。如果外部网口直接接入Internet，则可以将外部网口配置类型选定为静态或动态，视是否有固定IP而定。如果外部网口需要通过PPPoE协议拨号接入Internet，则需要指定网口类型为PPPoE，并设定用户名和密码。

网口类型说明如图8-11所示。

(1) 静态：为端口指定固定IP地址，可在“首要IP地址和网络掩码”处设置。

(2) 动态：动态获取IP地址，需要指定DHCP服务器IP地址。

(3) 桥接：无须设置IP地址，但需要指定“桥接至”某个网口(通常为外部网口)。

(4) PPPoE：指定拨号连接，需设定账户名和密码。

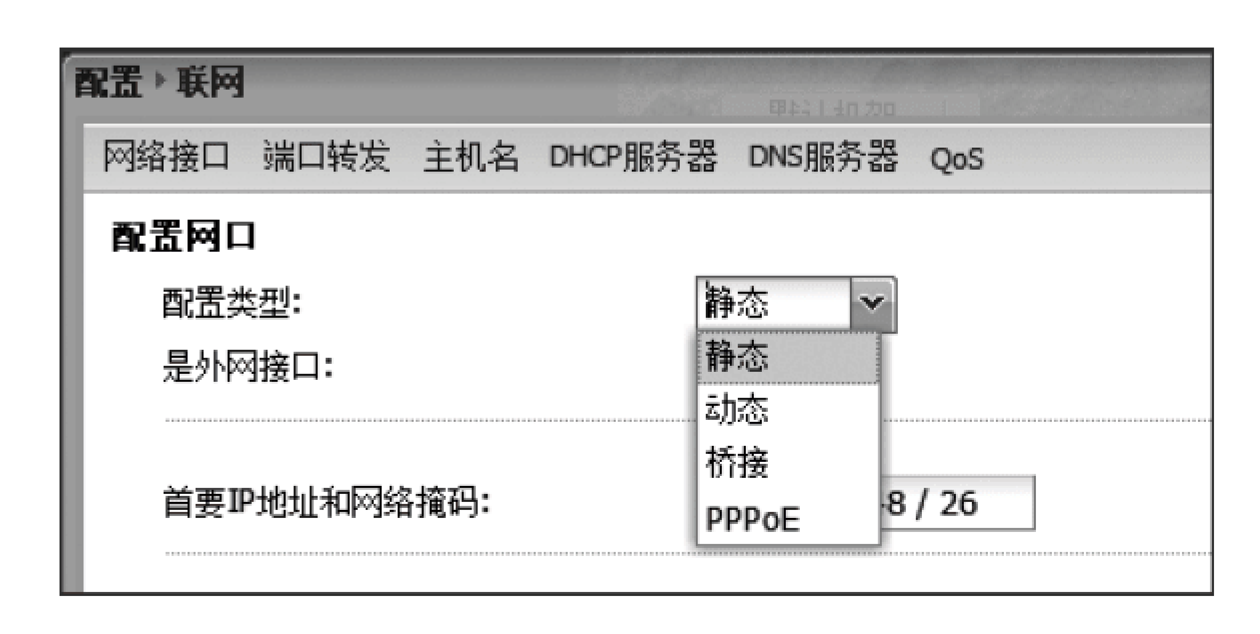


图8-11　配置网口类型

2) 内部网口配置

内部网口配置需要取消网口配置页面“是外网接口”复选框。内部网口必须配置为静态或桥接类型。

如果希望使用设备的路由功能，可以将内部网口设为静态类型，内网中其他设备会将该内网网口作为网关。此时可以在“DHCP服务器”选项卡中启用设备的DHCP服务，为内网的设备分配内网IP；内网中若有其他端口服务，则必须在“端口转发”选项卡中添加端口映射，以使外网用户可以正常访问内网服务。

如果设备前端已经提供了路由和DHCP服务，则可以将内部网口设置为桥接类型，桥接对象设为外部网口，且无须进行其他配置。

为了使设备正常工作，通常需要将eth0设为外部网口(External)，eth1设为内部网口(Internal)。外部网口和内部网口至少一个是静态的，以便顺利地通过Web方式进入UTM主界面。默认设置eth1的地址是192.168.2.1。

3) NAT策略设置

选择“配置”→“联网”→“网络接口”，单击“内部接口”前的“编辑”，在“NAT策略”区域单击“添加”，填写内部网络“地址和网络掩码”及“源地址”，最后单击“保存”按钮。

例如，假设UTM有一个内部接口，其IP地址和掩码为192.168.1.2/32，并且有两个外网地址1.2.3.1和1.2.3.2。如果希望将来自内网的数据包的源地址显示为1.2.3.2，那么就按照上述NAT策略配置将192.168.1.2/32与源地址1.2.3.2绑定。

在默认情况下，源地址是auto，意味着从内网发出来的数据包源地址显示为首要外网地址，即1.2.3.1。

2. 端口转发

端口转发通过设置规则对特定数据流进行转发。单击“端口转发”，进入配置页面，如

图 8-12 所示。选中复选框“开”,将使本条规则生效。单击每条规则后面的“编辑”按钮,可以对该规则进行编辑修改。

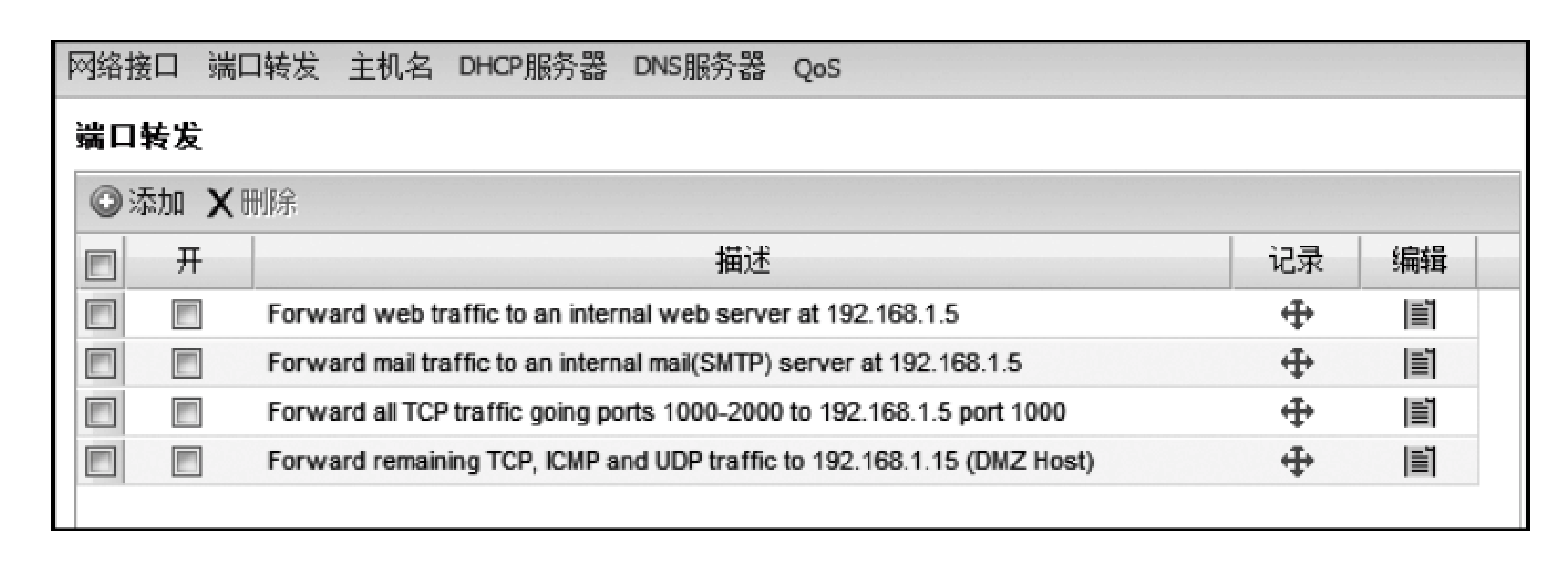


图 8-12 端口转发配置

单击“添加”按钮,可以设置新的转发规则。图 8-13 说明了如何生成一条新规则:将 Web 流量转发到内部 IP 地址为 192.168.1.5 的 Web 服务器。

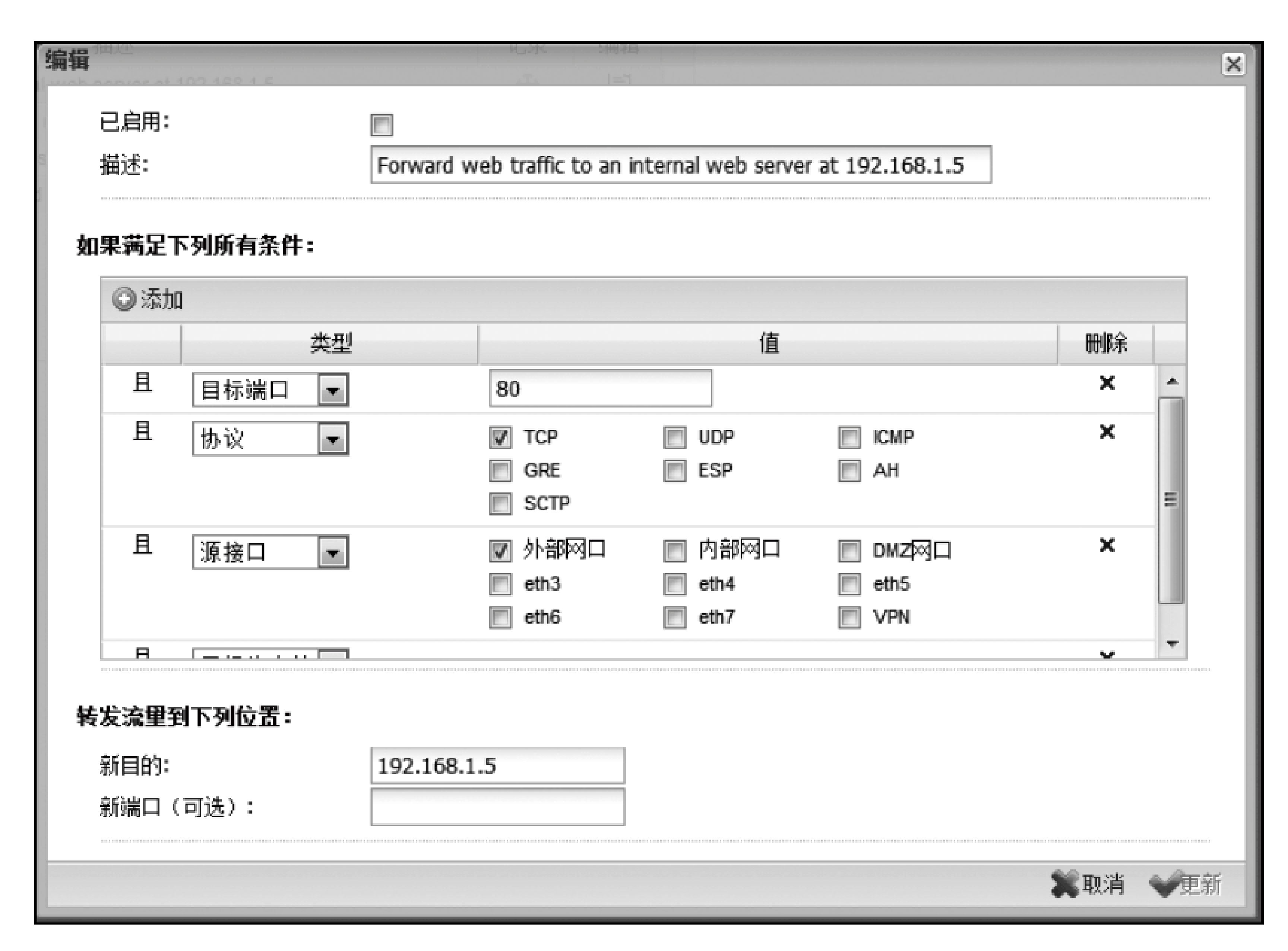


图 8-13 编辑新规则

3. 主机名

单击“主机名”,将进入设置页面,如图 8-14 所示。在此可以配置 UTM 服务器的主机名称,以及指定相应的 DNS 服务器。如果 UTM 服务器从 ISP 处获得了动态 IP 地址,那么可以通过此项的设置将该动态 IP 地址与你设定的静态域名绑定,从而方便通过 Internet 使用域名访问该 UTM。

“服务”后面的下拉列表中列出了提供动态 DNS 服务的提供商,可以进行服务注册。

4. DHCP 服务器

单击“DHCP 服务器”,进入设置页面,可以设置 DHCP 服务器,如图 8-15 所示。其功能和配置与一般 DHCP 服务器相同,可以为 UTM 所管辖的内部网络中的计算机分配动态 IP 地址。

当内网的某台计算机从 UTM 上获得了一个动态 IP 地址后,如果希望 VPN 客户端能

图 8-14　主机名设置

图 8-15　DHCP 服务器配置

够连接到那台计算机，那么就需要为其指定一个静态 IP 地址，这可以通过静态 DHCP 入口表的配置来实现：单击“添加”按钮，将生成一条新的表项，单击该表项任意字段即可进行设置，可以将特定“MAC 地址”与某“IP 地址”绑定，从而为该 MAC 地址的主机分配静态 IP 地址。

静态 DHCP 入口：列出所有已经被分配了静态 IP 地址的计算机。

当前 DHCP 入口：列出所有以及接受了 UTM 的 DHCP 服务的计算机。

5. DNS 服务器

UTM 可以为内网用户提供域名解析服务。单击“DNS 服务器”，进入设置页面，选中“已启用”复选框，则可启用 DNS 服务，如图 8-16 所示。

当内网某台计算机从 UTM 上获得动态 IP 地址后，UTM 将自动为该计算机增加一条 DNS 条目。当为内网某台计算机设置了域名解析功能后，内网的其他机器就可以通过域名访问该计算机。如果内网某台计算机没有设置域名，那么可以通过主机名和 IP 地址映射功



图 8-16 DNS 服务器配置

能实现使用主机名访问该计算机。

设置如下：在“静态 DNS 入口”项中单击“添加”按钮，将生成一条新记录，对该记录进行编辑，完成 IP 地址和主机名的绑定。

6. QoS

具体功能介绍和相关配置参见 9.5 节。

7. 高级选项设置

在默认安装情况下，UTM 将以标准配置方式安装，如果标准配置方式无法满足需要，用户可以通过“高级”选项进行设置，高级选项配置除包括了所有标准配置内容外，还增加了包过滤器、基本、旁路规则、ARP、路由、本地 DNS、DHCP 和 DNS、覆盖等配置项。高级配置要求管理员熟悉专业知识和经验。

可以直接单击“高级”进入高级配置模式向导，按照向导提示逐步进行配置，也可以打开“高级”下拉菜单选择需要配置的项，如图 8-17 所示。下面仅采用后者配置方式进行说明，并且因为“DNS 和 DHCP”配置与标准配置相同，将不再介绍，可以参见相关章节。

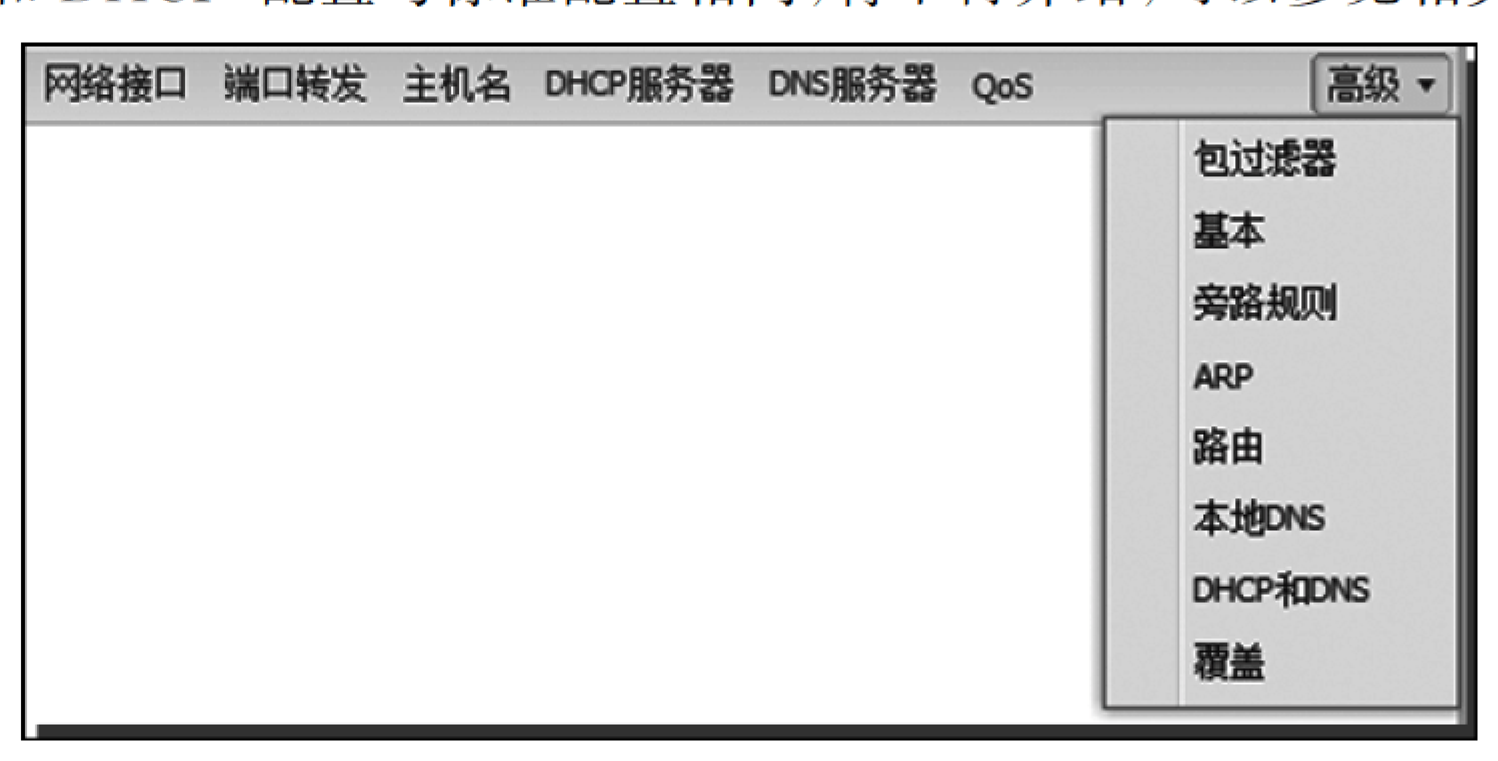


图 8-17 高级配置选项

1）包过滤器

虽然 UTM 提供了防火墙功能模块，但可以使用包过滤器满足特殊需求。使用包过滤器的前提是必须熟悉配置，否则请使用 UTM 的防火墙配置。而且，必须清楚对 UTM 是工作在传输层或网络层的，它将影响所有的 UTM 用户。

单击“高级”下拉菜单中的“包过滤器”，进入配置页面，如图 8-18 所示。

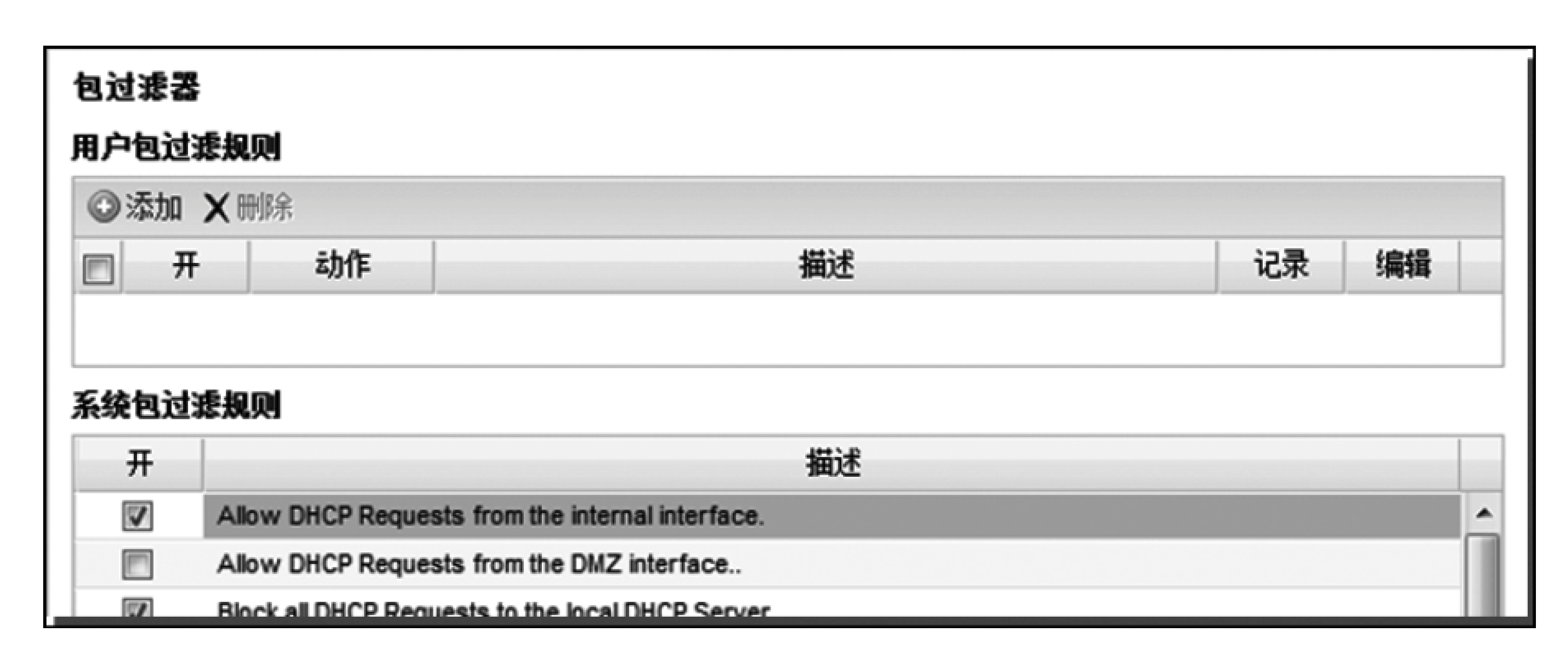


图 8-18　包过滤器配置

在“系统包过滤规则”表中列出了常用的过滤规则，可以通过勾选或清除复选框启动或禁用这些规则。

如果需要增加新的过滤规则，可单击“用户包过滤规则”中的“添加”按钮，将进入规则配置界面，如图 8-19 所示。

图 8-19　过滤规则配置

默认情况下，配置的这条规则是启用的。在“动作”后的下拉列表中选择以下需要的操作。

（1）通过：允许分组通过防火墙。

（2）丢弃：阻止分组通过防火墙，丢弃分组，且不给分组发送者发送任何消息。

（3）拒绝：丢弃分组，同时给分组的发送者返回一个 ICMP 报文“端口不可达”消息。

在“如果所有条件都满足”一栏中单击“添加”按钮，将生成一条新规则，在“类型”栏中可选择如下选项。

（1）源地址：源主机的 IP 地址。

（2）目的为本地：UTM 服务器上的任何外部接口或外部 IP 地址。

（3）目的地址：接收者的 IP 地址。

(4) 目标端口：接收者的端口

(5) 源端口：UTM 上第一个接收该流的端口。

(6) 源接口：UTM 上第一个接收该流的接口。

(7) 协议：流所使用的传输层或网络层协议。

(8) 源 MAC 地址：UTM 接口 MAC 地址。

设置完成后，单击“更新”使规则生效。

2) 基本

单击“高级”下拉菜单中的“基本”选项，进入配置页面，如图 8-20 所示。

可以启用或禁用下述功能。

(1) 发送 ICMP 重传包。

(2) 启用 SIP 工具。

(3) 转发端口管理。

(4) 仅 NAT-WAN 流量。

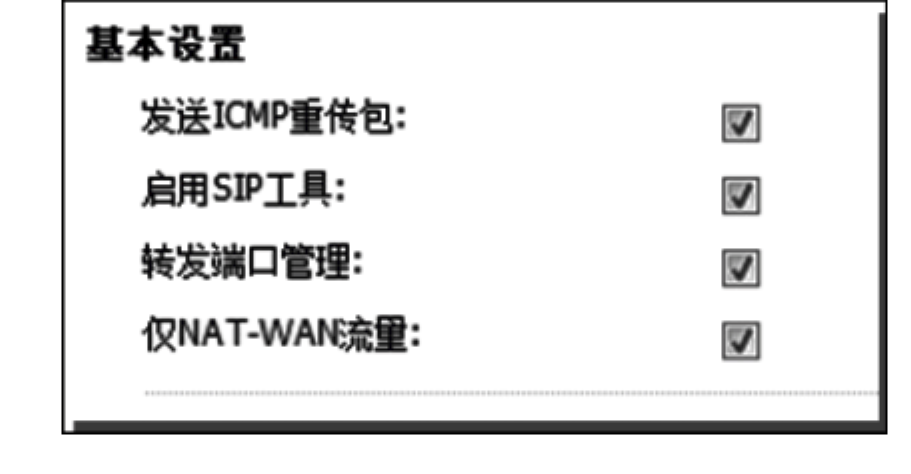


图 8-20　基本配置页面

3) 流量旁路配置

为了有效提高设备的可用性和可靠性，Untangle UTM 提供了流量旁路配置功能。用户可以根据实际需求，灵活地配置旁路多种不同的网络流量，如 VoIP、H323、RTP、SIP、DNS 等。

Untangle UTM 服务器架构如图 8-21 所示。

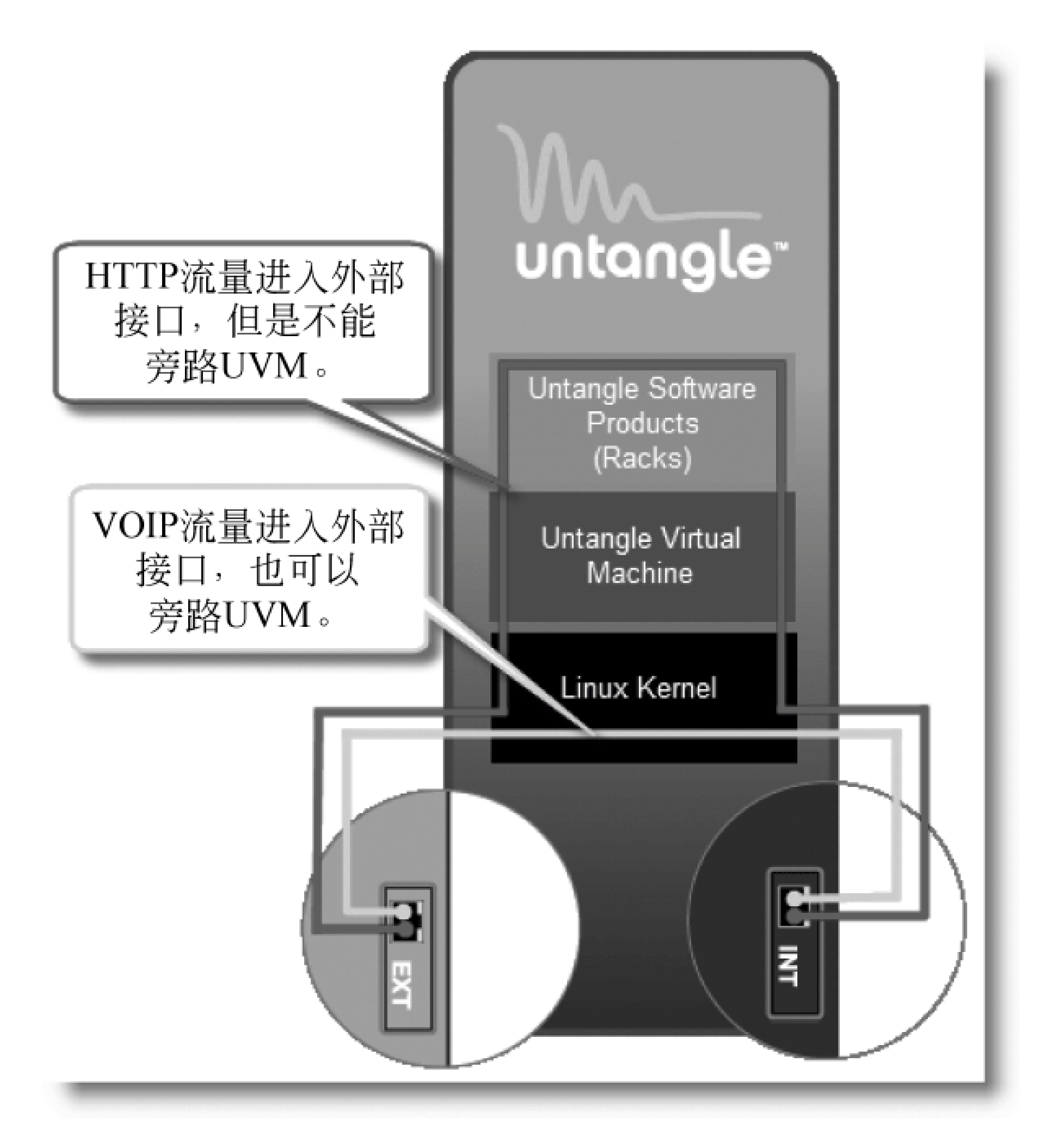


图 8-21　Untangle UTM 服务器架构

旁路规则(Bypass Rule)使得特殊的流量可以旁路过 UVM。Untangle Virtual Machine(UVM)是 Untangle Server 的进程，默认情况下，无论 Untangle 服务器是桥接还是网关模式，Untangle Sever 到达的流量总是通过 UVM，然后进入上层的 Rack 部分(安全功能组件)。匹配特定旁路规则的 Untangle 流量(如 VoIP)直接通过网卡，进入 Linux 内核，

而不必再通过 UVM。这是 Linux 内核来处理这些流量，而非 UVM。

当用户创建 Bypass 规则时，实际上是创建用户旁路规则。旁路规则分为用户旁路规则和系统旁路规则。系统旁路规则是默认预先配置于 Untangle 服务器中。在需要特殊处理和时间敏感性需求的情况下，旁路规则指导特殊流量旁路过 UVM，如 PPTP、IPsec VPN 协议等。

注意：

(1) 为了改进 QoS，启动 Untangle QoS 功能。旁路规则流量并不能旁路 Untangle QoS 系统(因为采用的是 Linux 内核的 QoS 支持功能，而非在 UVM 中处理)。如果旁路规则的目的是改善流量的时延和处理优先级，应当在 QoS 规则中增加一条规则，将该流量标记为高优先级。

(2) 协议控制模块是基于 UVM 的，协议控制模块的主要功能是实施流量策略。而旁路规则是指导特殊流量旁路过 UVM。

流量旁路具体设置如下：单击“配置”，进入“联网”选项。单击“高级”下拉菜单中“旁路规则”，进入配置页面，或者如图 8-22 所示。



图 8-22　旁路规则设置

通过添加旁路规则，可以将相应的流量进行直通旁路，这样便于网络的管理和维护，如添加旁路直通 VoIP 流量等。

4) ARP 管理

UTM 提供 IP 地址和 MAC 地址绑定功能，这样做有如下好处。

- 防止 ARP 欺骗。
- 避免因误配 IP 地址引起的混乱。

• 对某些不支持 ARP 协议的设备进行支持。

要增加一条“ARP 记录”，操作如下：单击“静态 ARP 入口”中的“添加”按钮，指定“IP 地址”和“MAC 地址”，如图 8-23 所示，而后单击“保存”按钮。



图 8-23　ARP 记录配置

5）路由

UTM 可以进行路由管理，单击“高级”下拉菜单中的“路由”选项，进入路由管理配置界面，如图 8-24 所示。

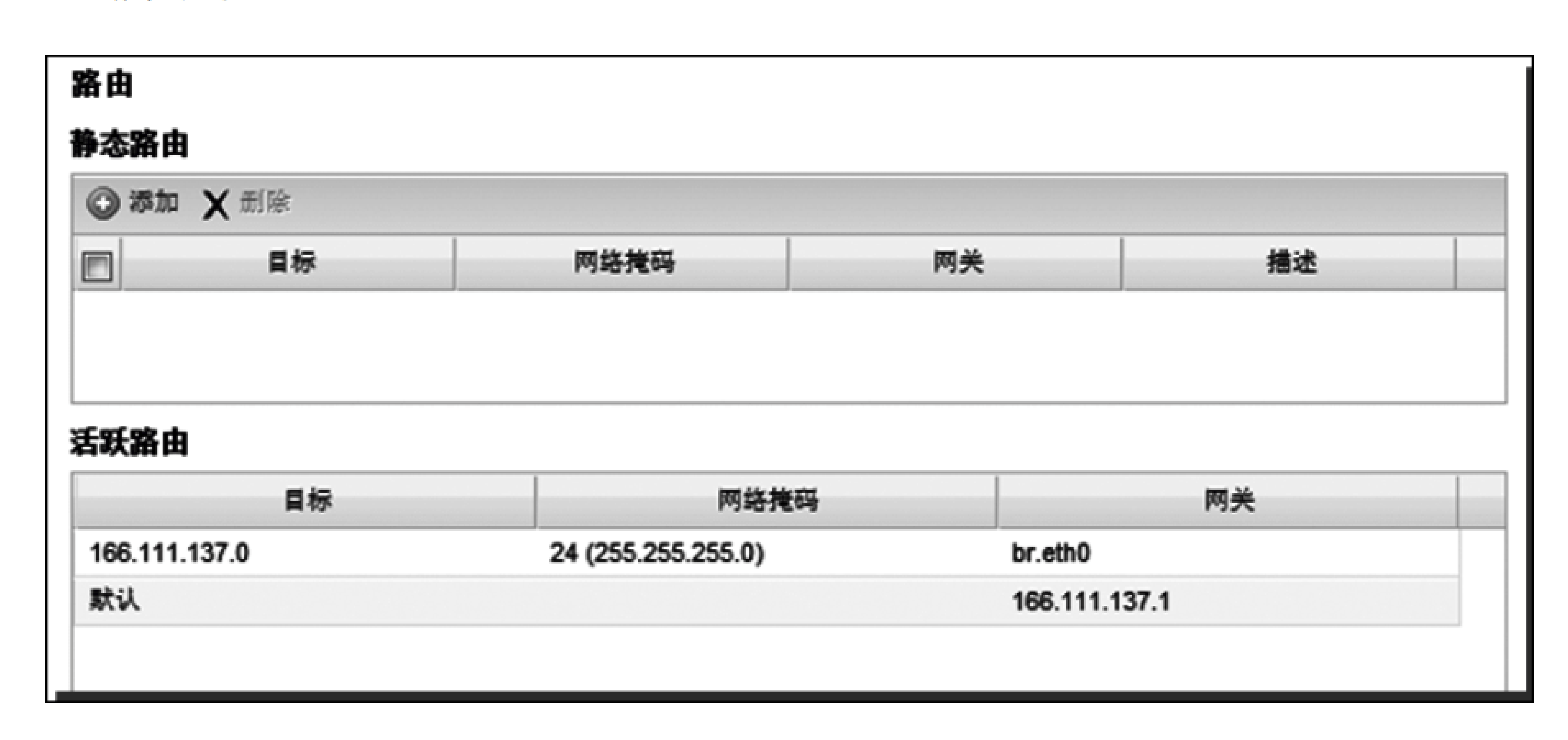


图 8-24　路由管理配置

（1）静态路由：显示手工添加的路由。

（2）活跃路由：显示当前活跃的路由表。

活跃路由是 UTM 决定将分组发往何处的依据。在配置了 UTM 及其接口后，将会在活跃路由表中自动增加一条默认记录，该默认记录是 Internet 的默认网关。

单击“静态路由”下的“添加”按钮，将增加一条新记录，可以对该记录进行编辑。

（1）目标/网络掩码：指定需要进行路由配置的网络。

（2）网关：指定接收来自路由网络数据的主机。

设置完成后，单击“保存”按钮

6）覆盖

文件覆盖为用户提供了防止特殊系统配置文件被改写的方法。一些专业用户习惯于手工编辑这些系统文件，这些文件包括系统配置文件和功能模块文件，前者是受 UTM 系统控制的，大多数可以被覆盖；后者则不受 UTM 系统控制，所以不能被覆盖。文件覆盖配置如图 8-25 所示。

覆盖

文件覆盖

添加 删除

	开	可写	文件路径	记录
☐	☑	☑	/etc/untangle-net-alpaca/sysctl	✥
☐	☑	☑	/etc/network/interfaces	✥
☐	☑	☑	/etc/hostname	✥
☐	☑	☑	/etc/mailname	✥
☐	☑	☑	/etc/resolv.conf	✥
☐	☑	☑	/etc/dnsmasq.conf	✥
☐	☑	☑	/etc/untangle-net-alpaca/dnsmasq-hosts	✥
☐	☑	☑	/etc/ddclient.conf	✥

图 8-25　文件覆盖配置

8.3.3　管理配置

管理配置页面可进行设备管理相关的属性设置，包括管理员账号的添加、删除及密码的修改；UTM 主界面的访问控制；本机证书的生成和管理；系统日志（Syslog）的配置等。

若内网接口不是静态类型，建议启用外部管理并将外部接口设为静态。

1．管理员账号

在主界面右侧栏中单击“配置”下面的“管理”按钮，进入管理配置界面，如图 8-26 所示。



图 8-26　Untangle UTM 管理配置

单击“添加”按钮，可以添加新的管理员，设置密码、邮箱等。同时也可以对现有管理员

的相关信息进行修改和删除。

（1）外部管理：对用户从外网访问 UTM 服务器操作进行设置。

（2）内部管理：对内网用户使用 HTTP 协议访问 UTM 服务器进行设置。

2．公共地址

如果 UTM 部署在路由器之后，UTM 需要知道该路由器的 IP 地址，因为一些应用在发送 URL 链接时需要知道外部地址。这里可以为 UTM 指定外部路由器的公共地址，必要时还可以指定一个端口进行转发或重定向数据。

当 UTM 拥有自己的域名，并且启用了动态域名解析服务时，就可以选择“使用主机名”，公共地址配置如图 8-27 所示。

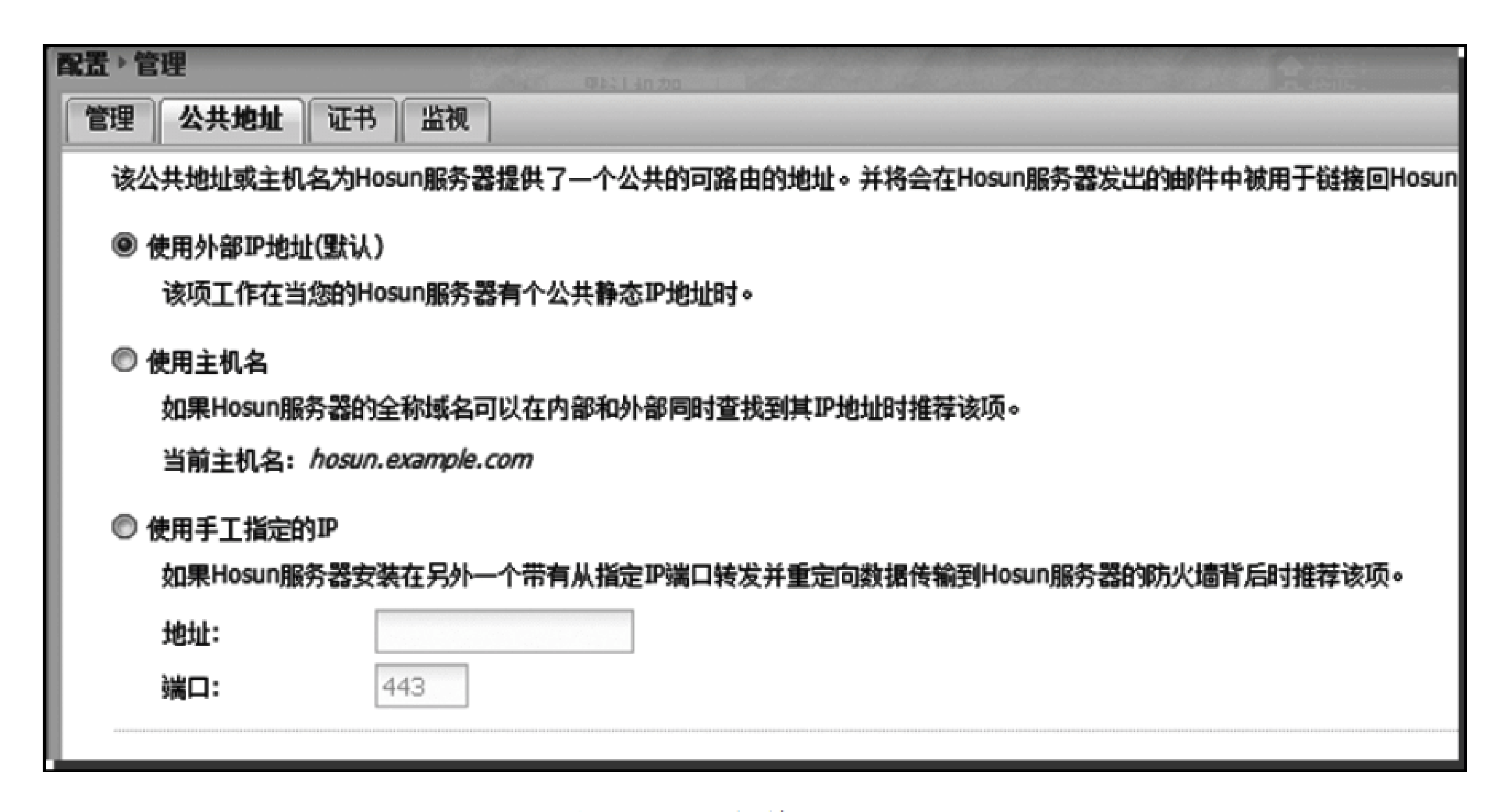


图 8-27　公共地址配置

3．证书

数字证书是 Web 服务器用来证明自己身份的凭证。当人们使用 SSL 协议通过浏览器访问 UTM 服务器时，UTM 服务器将向人们出示自己的证书，如果人们的浏览器没有将该证书加为受信任的证书，那么浏览器将发出“证书错误”的告警。UTM 通常是用自己生成的证书，因此在用户浏览器看来该证书是“未被授权的”，因此为了避免告警信息的干扰，请永久接受该证书。

证书配置如图 8-28 所示。要生成一个自签名证书，操作如下。

单击“创建一个自签名证书”，在弹出框中填写相关信息，而后单击“进行”。

创建一个证书签名请求：在申请如 Thawte、Verisign 等证书授权者的认证时，可通过复制并粘贴的方式使用该证书签名请求。

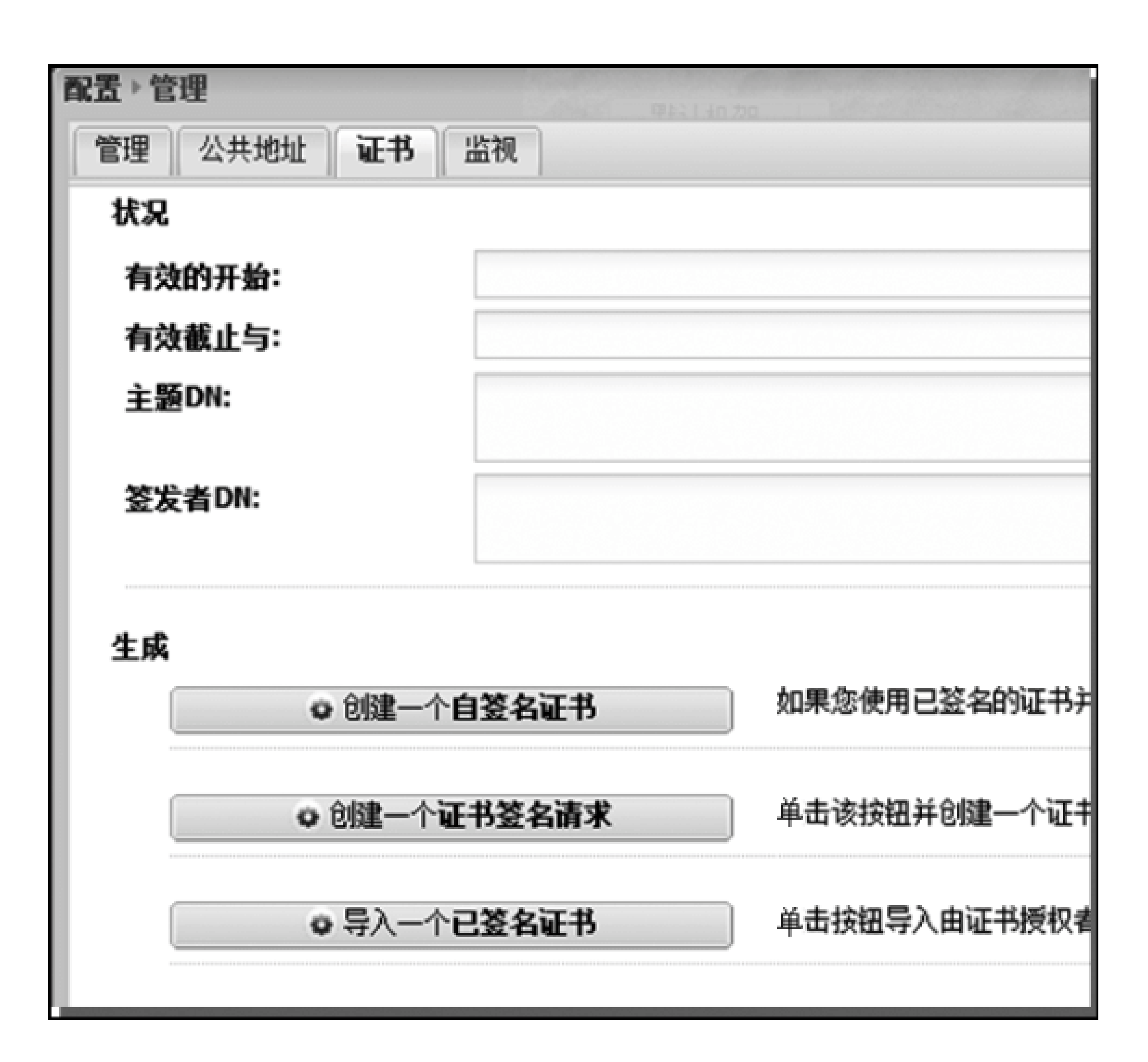
导入一个已签名证书：导入由证书授权者生成的基于上一步签名请求表的已签名证书。

4．监视

简单网络管理协议（SNMP）是知名网络管理规范。UTM 完全支持 SNMP 协议规范。在“监视”配置里，可以进行如下设置，如图 8-29 所示。

（1）禁用 SNMP 监视（默认设置）：禁用 SNMP 监视功能。

（2）启用 SNMP 监视：启用监视功能，需要设置下述 SNMP 参数。





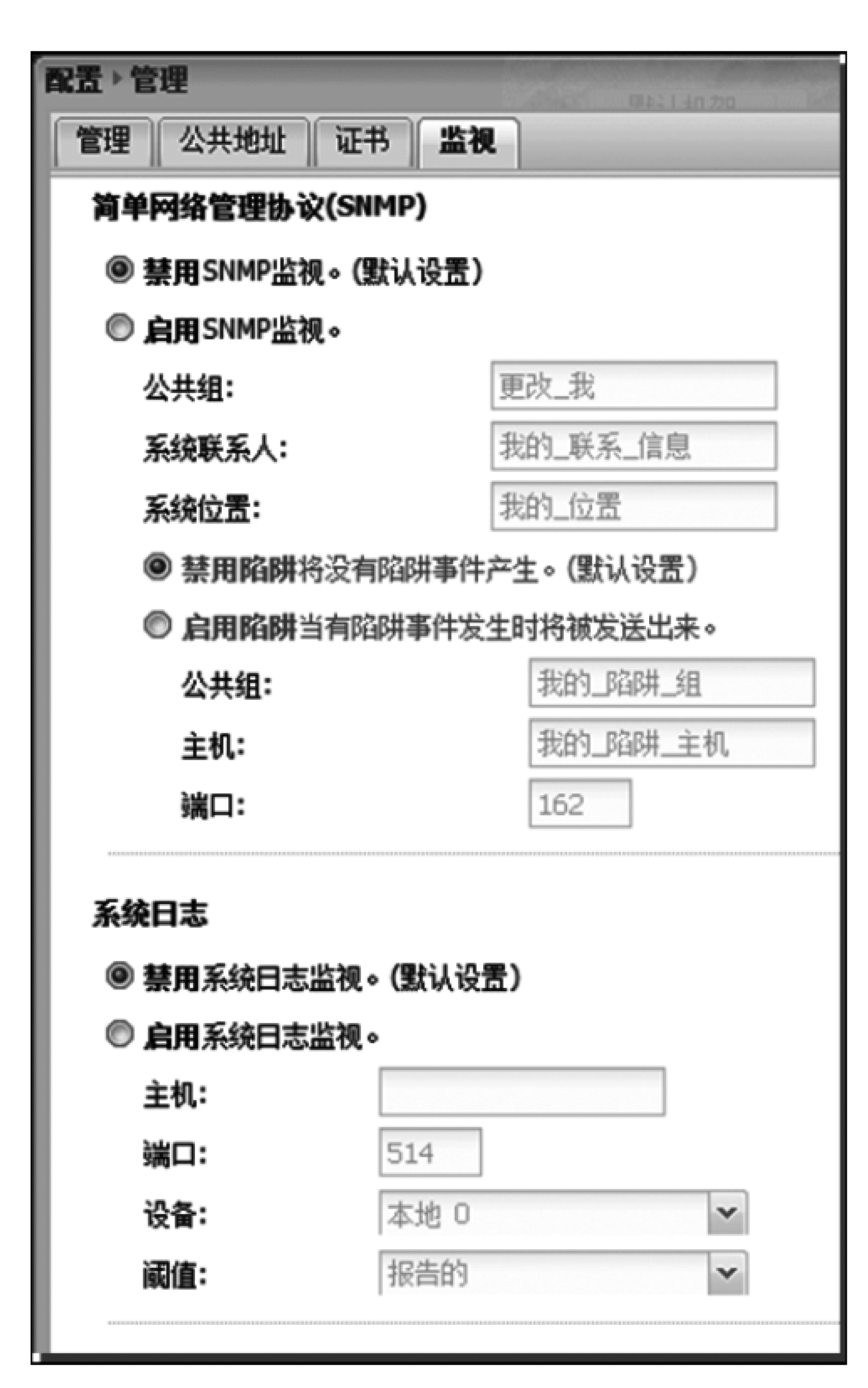


图 8-28　证书配置　　　　图 8-29　监视配置

① 公共组：设置 SNMP Community，默认为 Untangle。

② 系统联系人：接受 SNMP 消息的管理员邮箱地址。

③ 系统位置：对系统位置的描述。如果不知道请使用默认值。

④ 禁用陷阱：不产生 Trap 事件。

⑤ 启用陷阱：当有 Trap 事件发生，发送消息。注意：Trap 消息现在已经很少使用。

a. 公共组：验证 Trap 事件的 Community。

b. 主机：接收来自 UTM 统计信息的主机名或 IP 地址。

c. 端口：SNMP 的 Trap 端口为 162。

(3) 禁用系统日志监视：不产生系统日志。

(4) 启用系统日志监视：需要加载第三方消息发送模块，并设置下述参数。

① 主机：接收来自 UTM 系统日志的主机的主机名或 IP 地址。

② 端口：默认为 514，可以根据需要更改。

③ 设备：指定分配给 UTM 的区别于其他守护进程的设备。

④ 阈值：即优先级，表示消息内容的紧急程度。

8.3.4　电子邮件

单击“配置”下的“电子邮件”按钮，进入邮件配置页面。

1. 出站服务器

配置界面如图 8-30 所示。

配置 ‣ 电子邮件

出站服务器 | 安全发件人列表 | 隔离

出站电子邮件服务器(SMTP)

出站电子邮件服务器设置决定了Hosun服务器如何发送诸如报告、隔离摘要之类的电子邮件。大多数情况下，默认设置应该

直接发送邮件(默认)

通过指定的SMTP服务器发送邮件

服务器地址或主机名:

服务器端口: 25

使用认证

邮件发件人地址

Hosun服务器将从该地址发送邮件

hosun@example.com

电子邮件测试

电子邮件测试将按照现在的配置发送一封邮件到指定的地址。如果您没有收到测试电子邮件，您的设置就可能有问题。

电子邮件测试

图 8-30　出站服务器配置

该项配置较简单，按照界面提示配置即可。

2．安全发件人列表

安全列表是在垃圾邮件过滤器对合法邮件误报率比较高时采取的一种应对方法。Untangle UTM 的误报率并不高，因此一般不需要设定安全发件人列表，并且不允许管理员为每个用户都设定一个安全发件人列表，除非该用户有被隔离的合法邮件。

全局：为所有终端用户进行设置。

每用户：为个人用户进行设置。

安全发件人列表配置如图 8-31 所示。要配置“安全发件人列表”，进行如下操作：单击“安全发件人列表”，在“全局”表下单击“全局”选项，在弹出框中填写发件人邮件地址，如 zhansan@gmail. com 或 * @gmail. com，而后单击“保存”按钮。

配置 ‣ 电子邮件

出站服务器 | 安全发件人列表 | 隔离

全局

添加

电子邮件地址	删除

每用户

已选择净化

账户地址	安全列表大小	显示细节

图 8-31　安全发件人列表配置

3. 隔离

UTM 可以在隔离区存储可疑邮件,这样就可通过查看隔离区来避免邮件被误判。邮件是否被隔离是由软件功能模块决定的,比如垃圾邮件拦截等。有两类用户可以访问隔离区。

(1) 终端用户:UTM 所保护的网络中的用户。此类用户可以在隔离区清除或释放可疑邮件,但不能登录管理员界面。

(2) 管理员:拥有管理特权。

要进行隔离设置,请单击“隔离”选项卡,接入配置界面,如图 8-32 所示。

最大保留时间:指定保存多少天,过期会被删除。

发送每日隔离摘要:选中后将每天向用户自动发送 HTML 格式的隔离摘要信息,并通过链接指向它们的隔离信箱。

隔离摘要发送时间:修改发送摘要的时间。



图 8-32 邮件隔离配置

用户可以通过 https://UTMServerIP/quarantine/链接输入邮箱地址,手动请求发送隔离摘要。设置界面如图 8-33 所示。

请求隔离摘要邮件

输入您希望隔离摘要发送的电子邮件地址

电子邮件地址:

请求

图 8-33 手动配置要求发送隔离摘要

当将垃圾邮件拦截模块设置为扫描并隔离邮件时,它会对所有用户执行该操作,除非特别指定。如果将垃圾拦截模块设置为隔离所有用户的邮件,并且希望所有用户都能管理自己的隔离邮件,就不需要进行任何设置。但是,如果不希望垃圾邮件拦截模块隔离所有用户的邮件,而仅仅希望垃圾邮件拦截模块对某些用户的邮件进行标记,那么就需要进行如下

操作。

在“应隔离的地址”区域，删除包含通配符 * 的默认行，而后单击“添加”按钮，为需要隔离的用户创建表项，在表项中设定该用户的地址。这样对于不在表项中的用户，垃圾邮件拦截模块仅对它们进行标记。

如果不希望特定用户管理他们自己的隔离邮件或其订购的邮件分发列表隔离邮件，那么可以将这些用户或邮件分发列表的隔离邮件转发到一个专用邮件账号去。设置如下：在“隔离转发”区域单击“添加”按钮，在编辑对话框中进行如下设置。

（1）分发列表收件人邮件地址。

（2）接收地址。

单击“更新”按钮。

8.3.5 本地目录

在 UTM 上存储用户目录，包括登录、姓名、邮件和密码信息等，如图 8-34 所示。配置如下：

单击“配置”栏中“本地目录”项，在“本地用户”栏下单击“添加”按钮，输入相关信息，单击“更新”按钮。



图 8-34　本地目录配置

8.3.6 系统

1. 客户支持

“客户支持”区域如图 8-35 所示，可以选择或不选择如下选项。



图 8-35　客户支持配置

(1) 允许因客户支持目的安全的访问您的服务器：允许 Untangle 技术人员远程访问你的 UTM 服务器以解决可能的故障。

(2) 发送您的服务器数据用于客户支持目的：允许 UTM 服务器向 Untangle 技术支持发送服务器日志消息以解决可能的故障，UTM 不会发送敏感的网络信息。

在“手动重启”区域，可以选择“重启”服务器。

在“手动关闭”区域，可以选择“关闭”服务器。

2. 备份

Untangle 推荐人们对自己的配置进行备份，可以把当前的系统配置备份到本地计算机的一个文件以便于之后的系统恢复。备份文件的后缀名为.backup，再把当前的系统配置备份到一个文件后，将来可以用在“恢复”选项卡中来恢复配置，如图 8-36 所示。

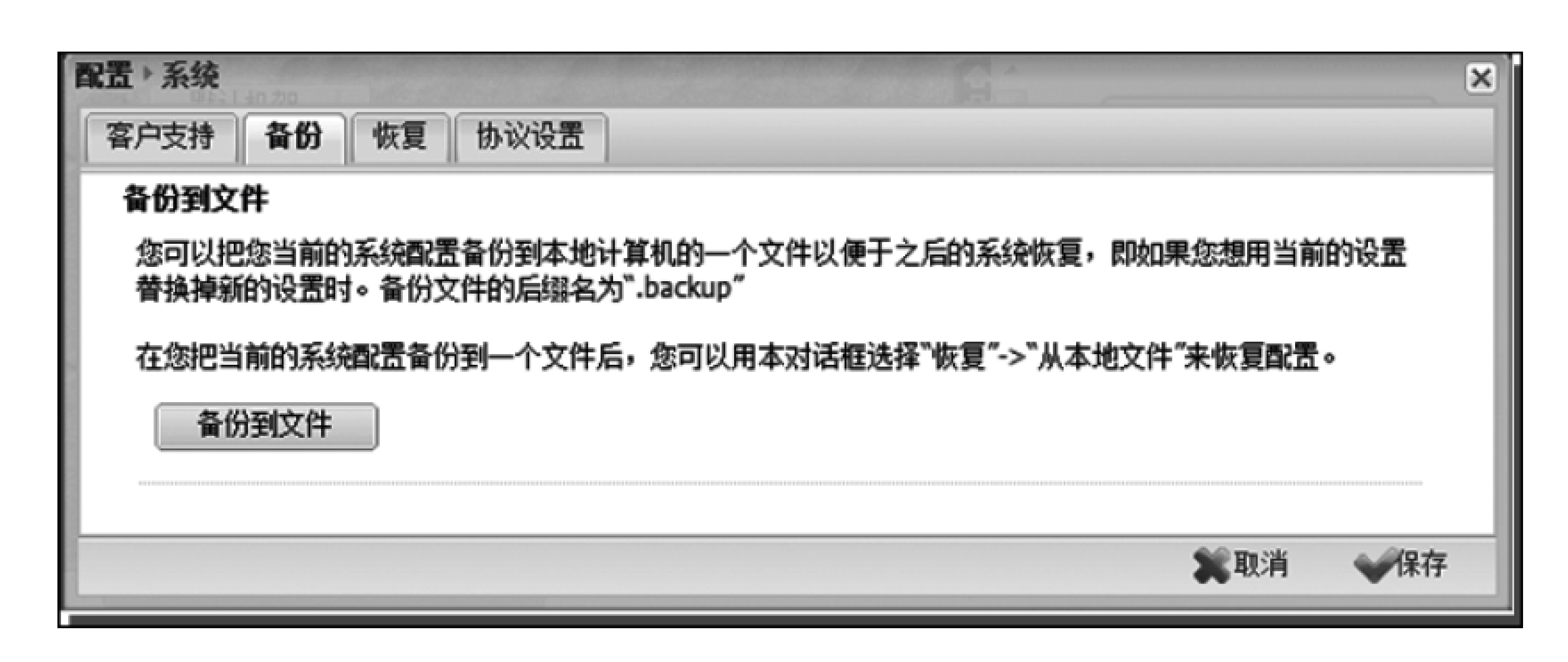


图 8-36　备份配置

3. 恢复

从本地计算机的备份文件中恢复一个以前的系统配置。备份文件的后缀名为.backup。恢复配置操作如下：单击“浏览”按钮，选择一个备份文件，单击“从文件恢复”按钮，再单击“保存”按钮，如图 8-37 所示。

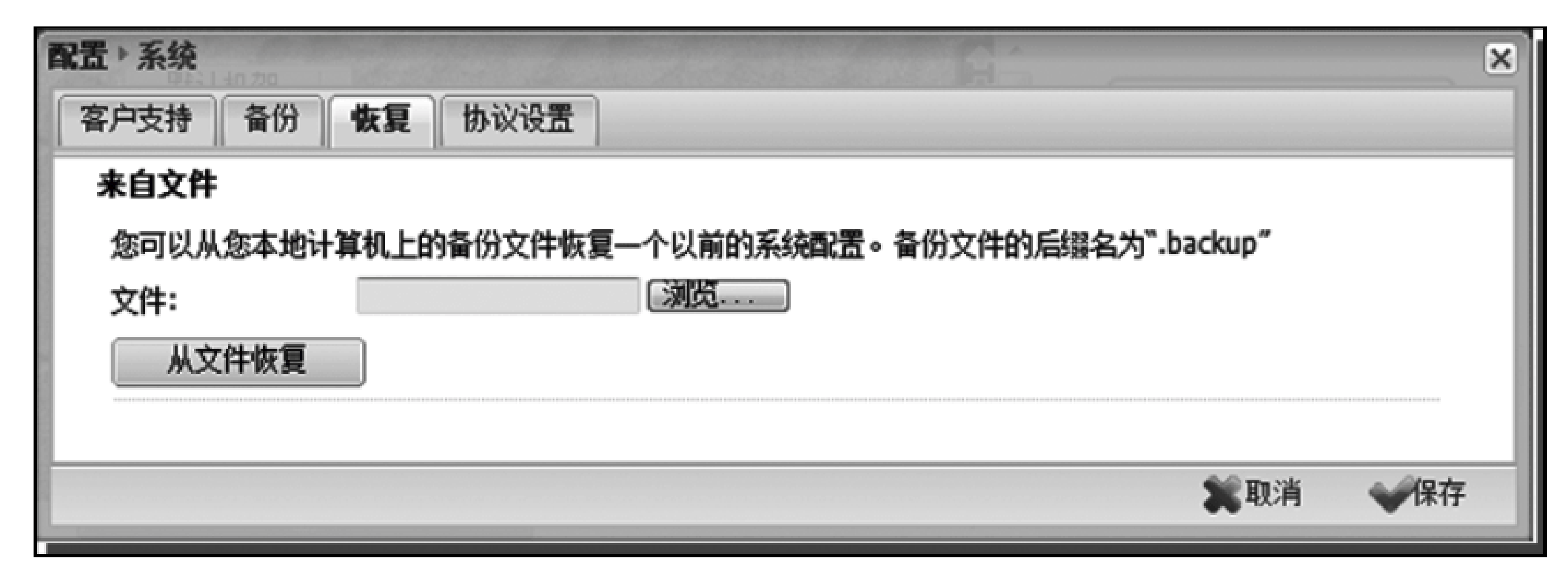


图 8-37　恢复系统配置

4. 协议设置

当进行故障排除时，可以通过禁用特定协议处理来排除 UTM 自身的问题。该项设置比较简单，只要选择禁用或启用相关协议处理功能即可，如图 8-38 所示。

8.3.7　系统信息

在系统信息中显示了系统的版本号，并为系统管理员提供了注册功能，如图 8-39 所示。



图 8-38　协议设置



图 8-39　系统信息显示

第 9 章　UTM 功用

单击每个功能插件后面的开关按钮，UTM 的每一项功能都可以关闭和打开。开关按钮下面的指示灯为绿色，表示此功能插件已开启；为黄色，表示插件关闭。

9.1　入侵保护

9.1.1　功能描述

入侵保护是一套入侵检测系统，它能够拦截所有流量、检测针对整个网络或者单一计算机的恶意行为。入侵保护采用特征检测的方法用来发现恶意行为，这种方法借鉴了已知的攻击模式数据库。入侵保护对于恶意行为的拦截并不会对系统性能产生影响，它对于恶意用户以外的其他用户来说是透明的。如果入侵保护检测出恶意行为，它将终止该行为的会话。入侵保护预设了合理的默认配置，因此入侵保护并不需要太多的定制，虽然用户可以更改这些默认值或添加自己的规则。

入侵保护提供了一个规则（特征）列表，可以对满足规则的行为进行阻挡、记录或者忽略。为了方便用户使用，UTM 评估这些规则和网络情况，在此基础上确定基于以下标准的合适默认值。

（1）如果规则一直都被视为阻挡恶意行为规则，入侵保护将默认地阻挡和记录。

（2）如果规则有时被视为阻挡恶意行为规则，入侵保护将默认地记录。

（3）如果规则从来不被视为阻挡恶意行为规则，入侵保护将默认地既不阻挡也不记录。

在大多数情况下，用户不必更改默认设置。如果规则阻止某个用户必须使用的软件应用程序流量，那么用户只需要禁用规则。

（1）如果用户启动一条阻挡规则，入侵保护将会阻挡与规则特征相匹配的流量。

（2）如果用户启动一条记录规则，入侵保护将会记录与规则特征相匹配的流量。

9.1.2　配置说明

在 Untangle UTM 主界面的机架中单击“入侵保护”选项，进入入侵保护的配置界面。

1. 状态

单击“状态”选项卡，页面中的信息解释如下。界面如图 9-1 所示。

（1）总的可用签名：入侵检测的所有可用规则数。

（2）总的签名记录：规则中动作为日志的规则数。

（3）总阻止的签名：规则中动作为阻止的规则数。

注意：对于每条规则，可以同时声明阻止和日志两个动作，也可以根据需要选择任意一个动作。

2. 规则

单击“规则”选项卡，显示入侵保护的规则列表，如图 9-2 所示。



图 9-1　入侵保护状态页面

默认机架 ▸ 入侵保护

状态 | 规则 | 事件日志

规则

添加

分类 ▲	阻止	日志	描述	标识	信息	编辑	删除
attack-responses	☐	☑	Potentially Bad Traffic (index of /cgi-bin/ response)	1666	no info		×
attack-responses	☑	☑	Successful Administrator Privilege Gain (Microsoft cmd	2123	no info		×
attack-responses	☐	☑	Potentially Bad Traffic (command error)	495	no info		×
attack-responses	☐	☑	Potentially Bad Traffic (oracle one hour install)	1464	no info		×
attack-responses	☐	☑	Attempted Information Leak (403 Forbidden)	1201	no info		×
attack-responses	☐	☑	Potentially Bad Traffic (command completed)	494	no info		×
attack-responses	☑	☑	Successful User Privilege Gain (successful cross site	2412	no info		×
attack-responses	☑	☑	Successful Administrator Privilege Gain (successful go	1810	no info		×
attack-responses	☑	☑	Successful Administrator Privilege Gain (successful ka	1901	info		×
attack-responses	☐	☑	Potentially Bad Traffic (directory listing)	1292	no info		×
attack-responses	☑	☑	Unsuccessful User Privilege Gain (rexec username too	2104	no info		×
attack-responses	☐	☑	Misc Attack (successful gobbles ssh exploit uname)	1811	no info		×

页 1 页共 99 页　　正在显示主题1 - 在2467中的第25个主题

变量

添加

名称 ▲	通过	描述	编辑	删除
$AIM_SERVERS	[64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,6	Addresses of possible AOL Instant Messaging		×
$HTTP_PORTS	80	Port that HTTP servers run on		×
$HTTP_SERVERS	$HOME_NET	Addresses of possible local HTTP servers		×
$ORACLE_PORTS	1521	Port that Oracle servers run on		×
$SMTP_SERVERS	$HOME_NET	Addresses of possible local SMTP servers		×
$SQL_SERVERS	!any	Addresses of local SQL servers		×
$SSH_PORTS	22	Port that SSH servers run on		×
$TELNET_SERVERS	$HOME_NET	Addresses of possible local telnet servers		×

移除　　取消　保存

图 9-2　入侵保护规则页面

其中规则的含义如下所示。

(1) 分类：根据保护的内容，把所有的入侵保护规则分为多个大类，如 web-iis、web-php 等。

(2) 阻止：规则的动作，阻止所有匹配此条规则的网流。

(3）日志：规则的动作，记录所有匹配此条规则的网流。

(4）描述：规则描述。

变量中配置在规则中使用的全局变量。

单击“添加”、“编辑”和“删除”按钮可以分别添加新规则、更新规则和删除规则。其编辑界面如图 9-3 所示。在签名中输入规则的表达式，用以匹配网流。变量的添加、编辑与此类似。

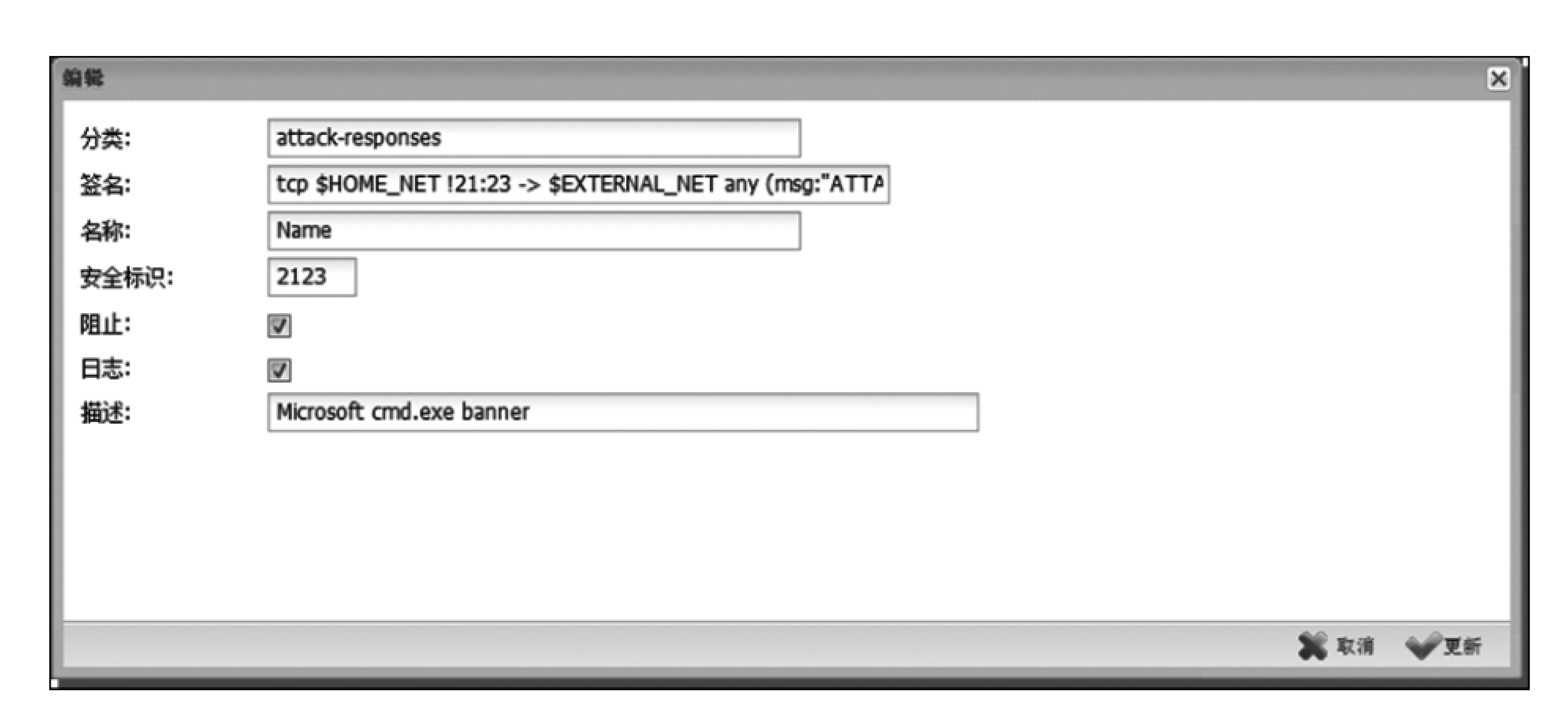


图 9-3　入侵保护规则编辑页面

3. 事件日志

单击“事件日志”选项卡显示入侵保护的规则列表，如图 9-4 所示，其显示所有匹配动作为日志的网流。



图 9-4　入侵保护日志页面

注意：入侵保护的规则能够通过网络自动更新，用户一般采用默认配置即可。

9.2 防 病 毒

9.2.1 功能描述

Untangle UTM 防病毒模块基于一个开源的病毒扫描器——AntiVirus。防病毒模块具有高速和准确的特点。

防病毒模块具有以下功能。

(1) 检测病毒、蠕虫和木马。

(2) 扫描压缩文件，如 ZIP、RAR、Tar、Gzip、Bzip、MS OLE2、MS Cabinet 文件、MS CHM 和 MS SZDD 等。

9.2.2 具体配置

在 Untangle UTM 主界面的机架中单击“防病毒”选项，进入“防病毒”的配置界面。

1. 针对 Web 的病毒扫描

单击互联网(Web)选项卡，其界面如图 9-5 所示。清除复选框或选中复选框可以禁用/启用以下设置。

(1) 扫描 HTTP：是否要扫描 HTTP 协议。

(2) 文件扩展名：配置扫描 HTTP 协议的文件类型。用户可以自行添加和删除。

(3) MIME 类型：配置扫描的 MIME 类型，用户可以自行添加和删除。

(4) HTTP 协议允许断点续传功能：当此功能开启时，Untangle UTM 将无法对传输的文件进行病毒扫描。当 HTTP 进行断点续传时，包含病毒的文件可以在多次连接中进行传输，而 Untangle UTM 的服务器传输只能看到部分文件，而不能执行完整扫描。

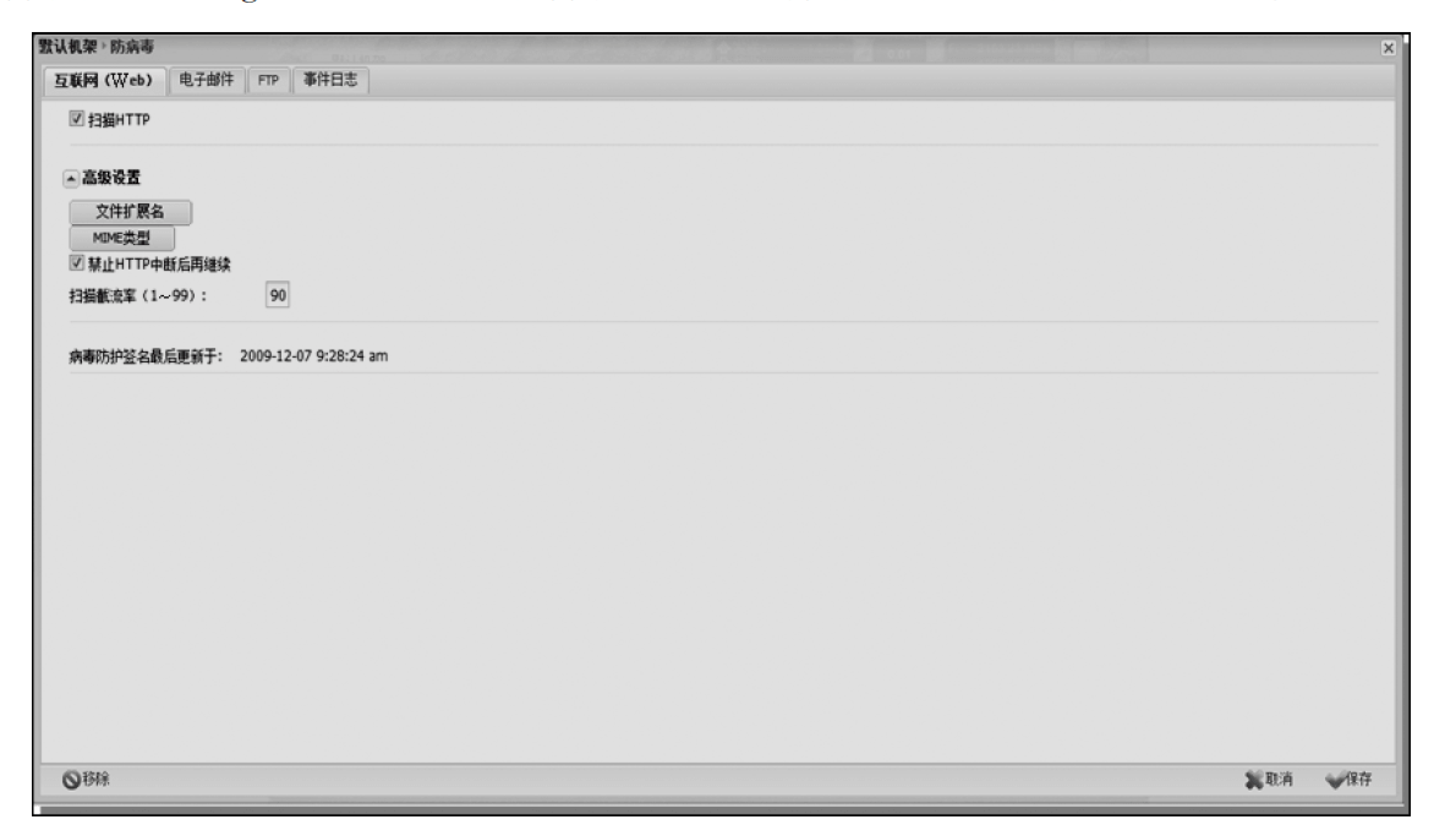


图 9-5 防病毒组件“互联网”选项卡

2. 针对电子邮件的病毒扫描

单击“电子邮件”选项卡，如图 9-6 所示。根据配置需求，选择扫描的 E-mail 协议类型，协议可以选择的动作如下。

（1）扫描 SMTP/POP3/IMAP：当该复选框被选中，Untangle UTM 将对两个方向上的电子邮件进行病毒扫描，除非有自定义策略来覆盖这些策略。

（2）删除感染：删除感染的电子邮件。

（3）允许通过：放行所有的电子邮件。

（4）阻止邮件：阻断包含病毒的 E-mail 网流。

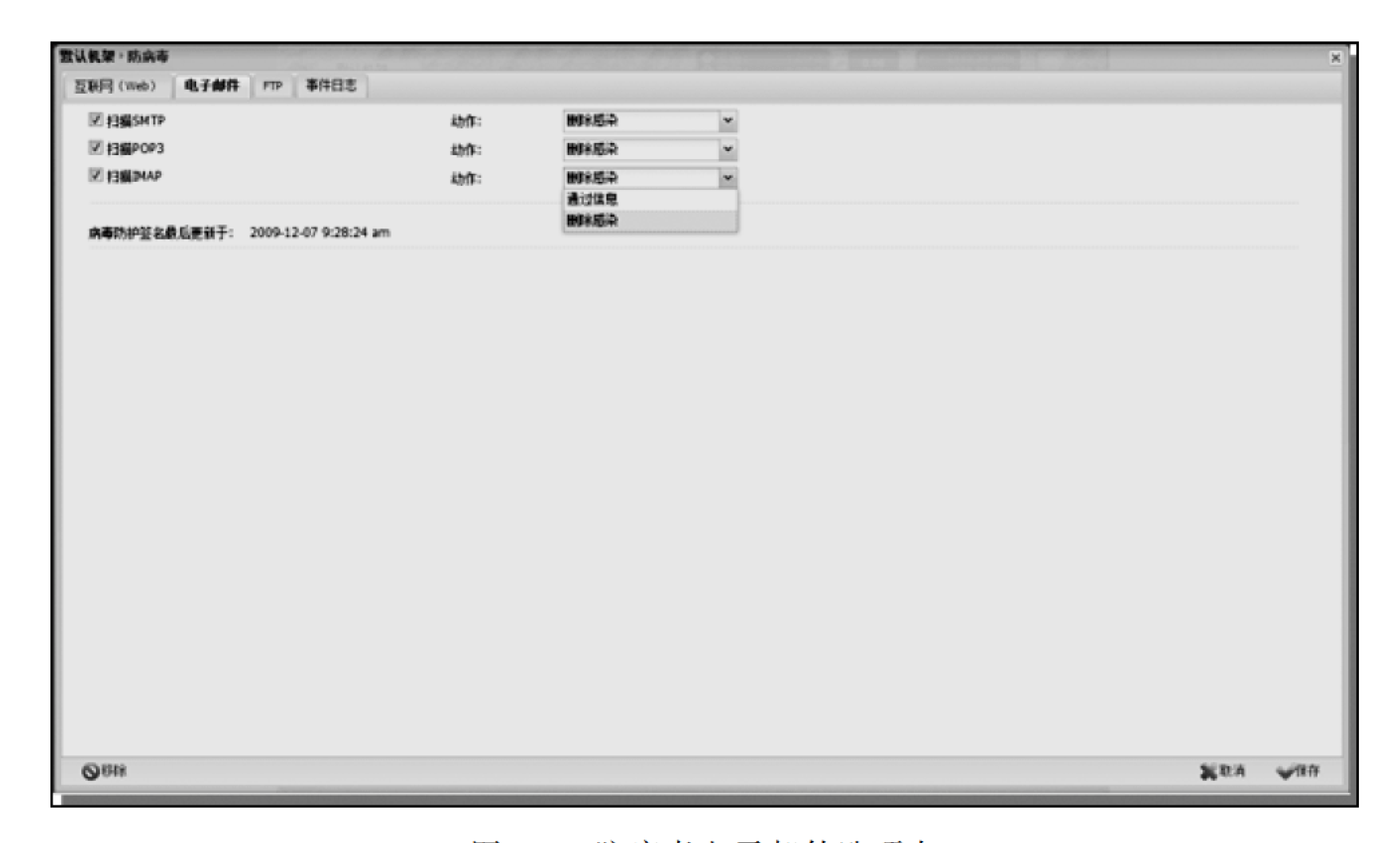


图 9-6　防病毒电子邮件选项卡

3. 针对 FTP 的病毒扫描

单击 FTP 选项卡，如图 9-7 所示。根据配置需求，清除复选框或选中复选框禁用/启用以下设置。



图 9-7　防病毒 FTP 选项卡

（1）扫描 FTP：是否扫描 FTP。因为当允许 FTP 下载断点续传允许的话，包含病毒的文件可能通过多个连接进行传输，而 Untangle UTM 只能对文件的部分内容进行病毒扫描，而不是对整个文件进行完整地扫描。

（2）扫描速率（百分比）：这是一项高级功能，用以控制文件相对于扫描的下载速率。作为一个高级功能，用户不应该改变这个选项，除非在 Untangle UTM 的技术支持下。

4. 事件日志

单击"事件日志"选项卡显示扫病毒的日志信息，如图 9-8 所示。可以在左下角选择框中选择显示类型，包括所有事件、被感染事件、HTTP 事件、FTP 事件、SMTP/IMAP 事件和 POP3 事件。



图 9-8 "事件日志"选项卡

9.3 防火墙

9.3.1 功能描述

Untangle UTM 防火墙组件提供传统的防火墙功能，基于规则进行阻断和记录。尽管"防火墙"一词已经发展到包括许多功能，但 Untangle UTM 防火墙仍是用于阻断流量的传统防火墙。其他功能如端口转发或协议拦截，将由其他组件提供。

Untangle UTM 防火墙使用简单的规则列表来判定是否拦截或记录的流量。每次防火墙检测到会话时，列表中的规则将被逐一匹配。防火墙将采纳第一个匹配成功的规则对应的动作。

防火墙规则以下列条件组合为基础。

（1）流量类型（传输层协议）。

（2）源接口。

（3）目标接口。

（4）源地址。

（5）目标地址。

（6）源端口。

（7）目标端口。

9.3.2 具体配置

在 UTM 主界面的机架中单击防火墙的配置，进入防火墙的配置界面。

1. 规则

单击“规则”选项卡，界面如图 9-9 所示，显示当前的所有规则。

（1）启用：规则是否启用。

（2）描述：规则的内容描述。



图 9-9　防火墙规则页面

可以添加、编辑和删除规则。防火墙规则编辑页面如图 9-10 所示。

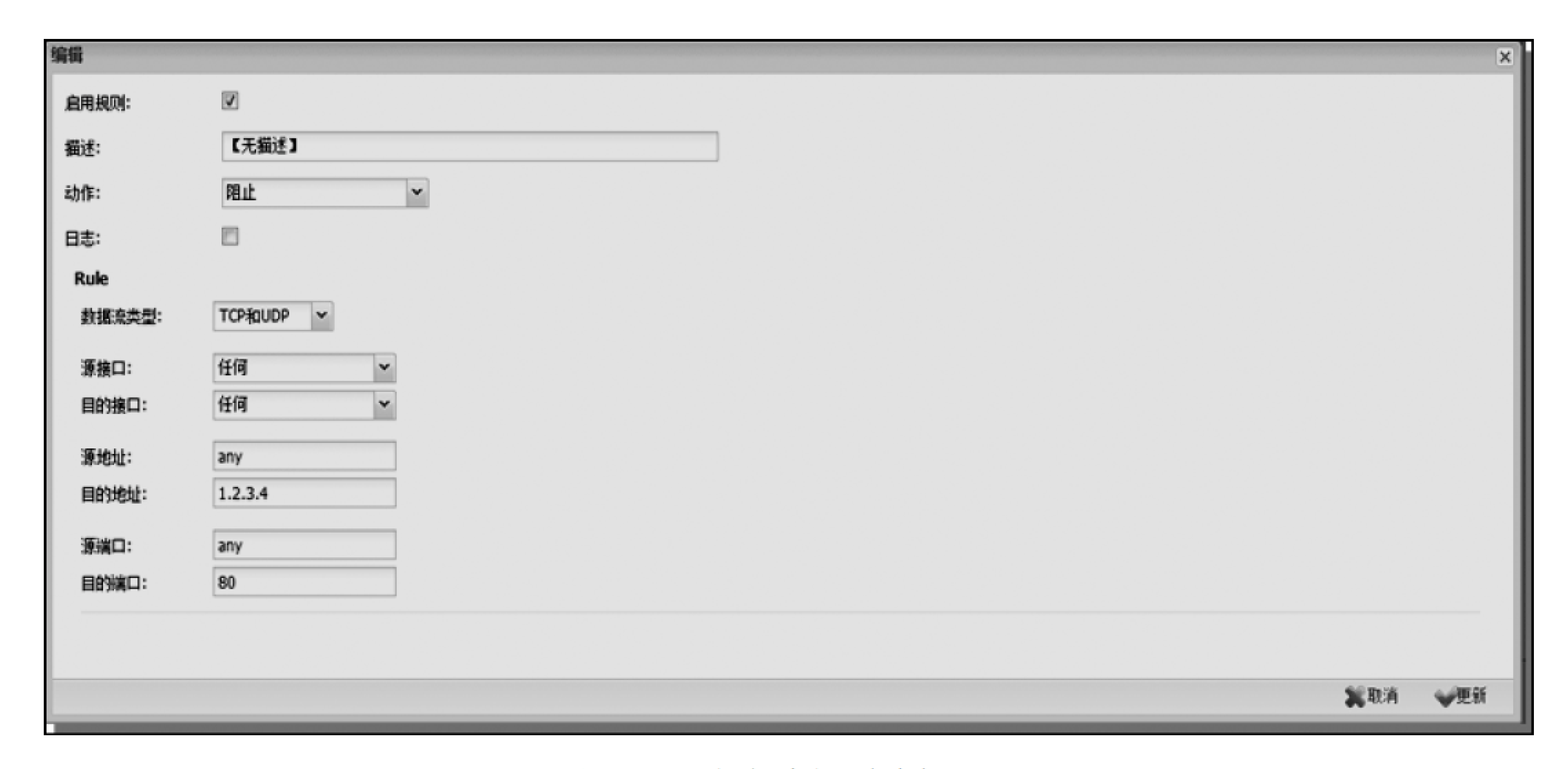


图 9-10　防火墙规则编辑页面

图 9-10 中各字段含义如下。

(1) 启用规则：规则是否启用，如果启用则勾选。

(2) 描述：规则描述。

(3) 动作：通过或者阻止。

(4) 日志：如果要记录则勾选。

(5) 数据流类型：UDP、TCP、UDP 和 TCP、所有流。

(6) 源接口和目的接口：网流的设备接口。

(7) 源地址和目的地址：网流地址。

(8) 源端口和目的端口：网流端口。

2. 使用防火墙

网络管理员可以根据需要构建防火墙规则，默认为所有会话都是通过防火墙的。

当 Untangle UTM 被配置成为 NAT 网关设备，或者以透明模式接入到其他 NAT 网关之后时，所有除端口转发的规定会话以外的连入会话都将被阻断。此时，防火墙组件就不必作用于连入会话，但仍可作用于连出会话。

表 9-1 是 Untangle UTM 防火墙的几种常见应用场景。

表 9-1 Untangle UTM 防火墙的几种常见应用场景

方　　案	描　　述	设　　置
阻断除特定流量以外的所有流量	本场景中，除了特定的配置为允许的流量，所有流量都将被阻断。这是最安全的应用方式，但也要求更多维护，也更有可能带来问题	将默认动作(最后一条规则)设置为阻断，然后添加指定允许的规则。通常需要添加允许的规则包括 DNS、网页和其他协议等连出会话规则，以及需要的其他连入会话规则
关闭特定的端口	本场景中，大多数流量是通过的，但防火墙将用于明确阻止特定流量类型通过某些端口或机器。这是一种常用方案，因为它在安全和维护便利中取了折中	将默认动作(最后一条规则)设置为通过，为特定要阻断流量创建规则

3. 防火墙规则分析

防火墙以规则为基础匹配会话和连接。当检测到新的会话时，防火墙将逐一匹配规则，当会话满足规则中的所有条件时，规则匹配成功。防火墙将应用最先匹配成功的规则所对应的动作，如果没有规则匹配成功，防火墙将采用默认规则。

每条规则将比较流量类型(传输层协议)、源接口和目标接口、源地址和目标地址以及源端口和目标端口，具体可参阅表 9-2。

表 9-2 比较的名称及描述

比较的名称	描　　述
流量类型	可选择类型包括 TCP、UDP、TCP 和 UDP、任意
源接口	发起会话请求的一端的接口，可以选择外部接口、内部接口、DMZ、虚拟专用网(VPN)等
目标接口	响应会话请求一端的接口，可选同上
源地址	发起会话请求一方的 IP 地址，IP 地址分为五类，“任意”表示任意地址，1.2.3.4 表示单一 IP 地址，“1.2.3.4，1.2.3.5，1.2.3.6”表示多个 IP 地址，1.2.3.4～1.2.3.100 表示某 IP 地址段，1.2.3.4/24 表示某子网内的 IP

续表

比较的名称	描　　述
目标地址	响应会话请求一方的 IP 地址，书写规则同上
源端口	发起会话请求一方的端口，端口分为四类，“任意”表示任意端口，80 表示单一端口，“80，81，82”表示多个端口，80～89 表示端口段
目标端口	响应会话请求一方的端口，书写规则同上

注意：防火墙只匹配连接，而不是匹配报文。如果连接判定为通过，该连接中的所有报文将自动通过，而不会再进行匹配。

4. 构建规则集

对于每个新连接，规则列表都是从第一条开始匹配，直到匹配成功。这使管理员可以小心地安排规则顺序以创建详细的防火墙策略。例如，第一条规则是允许来自 192.168.1.10 的 SSH 流量，第二条规则是阻断所有 SSH 流量，那么该规则集的结果就为：除了来自 192.168.1.10 的可以通过外，其他的 SSH 连接全部被阻断。

注意：可以通过上下拖曳来调整规则的先后顺序。

5. 事件日志

单击“事件日志”选项卡显示扫病毒的日志信息，如图 9-11 所示。



图 9-11　防火墙日志页面

9.4　协议控制

9.4.1　功能描述

该模块用于应用层流量管理，特别是针对 P2P 应用、游戏的识别与控制。到目前为止，已经支持实际的主流协议达 100 种以上。其在精确识别协议，即对应用分类的基础上，根据

用户自定义策略，提供灵活方便的协议控制管理机制，如用户能够根据需求自行添加协议规则，记录或者阻止协议。

9.4.2 具体配置

1. 阻挡或者记录协议

协议控制通过设置能够阻断指定的协议，而不管其端口的变化。系统默认的配置是放行所有协议，仅记录即时通信协议(IM)。协议控制通过特征码能够在所有端口上识别协议。当前很多协议通过端口阻断(如 IM、P2P 等)难以通过传统的防火墙，因为它们大多采用了端口跳转技术，即当客户端在其默认端口多次与服务器连接失败后，其将转至连接 80 或者 25 端口。这两个端口一般用于 HTTP 和 E-mail 协议，是不能阻断的。而本系统能够有效识别协议的端口跳转，成功阻断和记录该协议的每个连接。

用户可以按照需求配置阻断特定协议的流量，包括进入或连出保护网络的所有流量。系统默认配置中，列出了常见的 100 多种协议，如图 9-12 所示。用户也可以对某种协议采用记录的方式，而该协议信息的日志将可在报表中审阅。如果用户需要查看网络内部是否有人在使用文件传输(FTP)协议，对 FTP 协议配置为记录，然后查看日志即可。

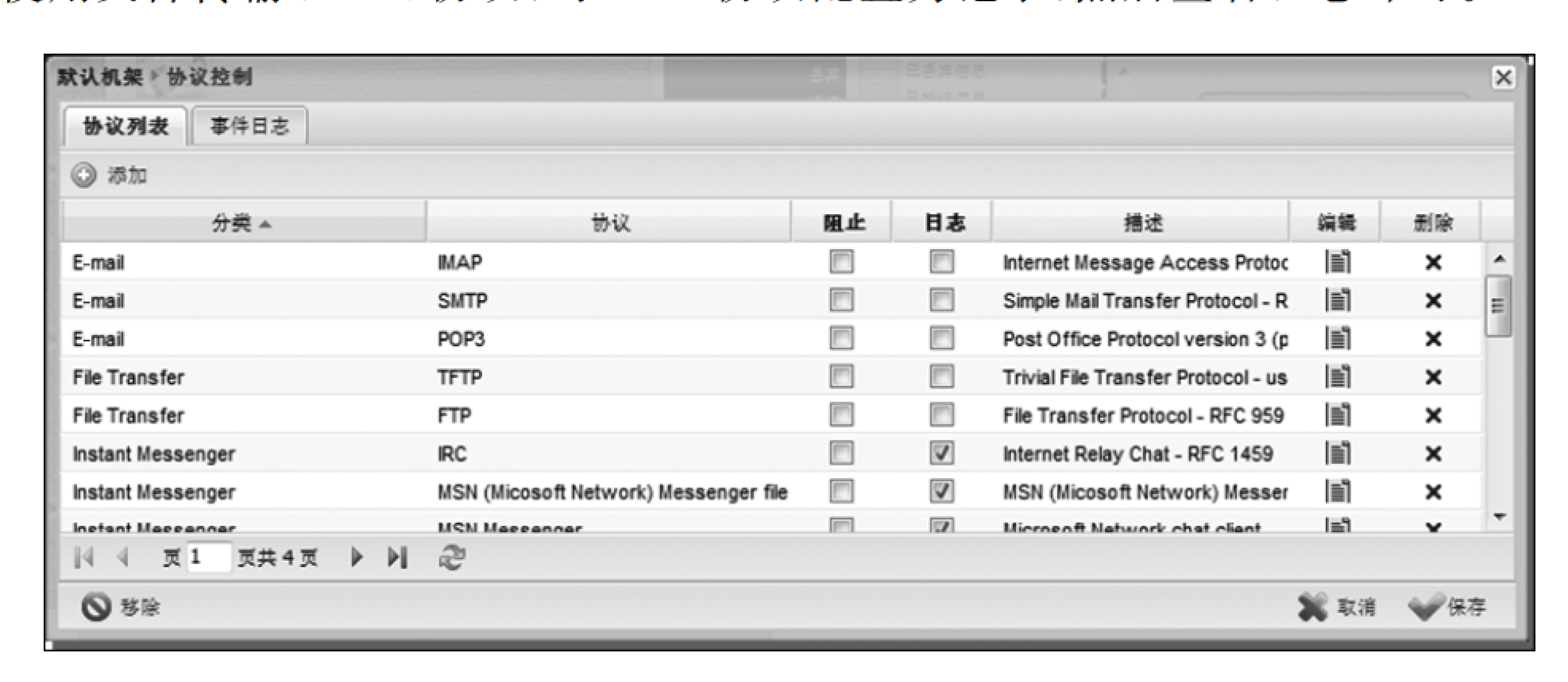


图 9-12 协议控制配置

2. 用户自定义协议

如果系统中并不支持用户想控制的某种协议，用户可以通过界面自行添加协议特征，进行阻断或者记录。用户自定义协议控制如图 9-13 所示。

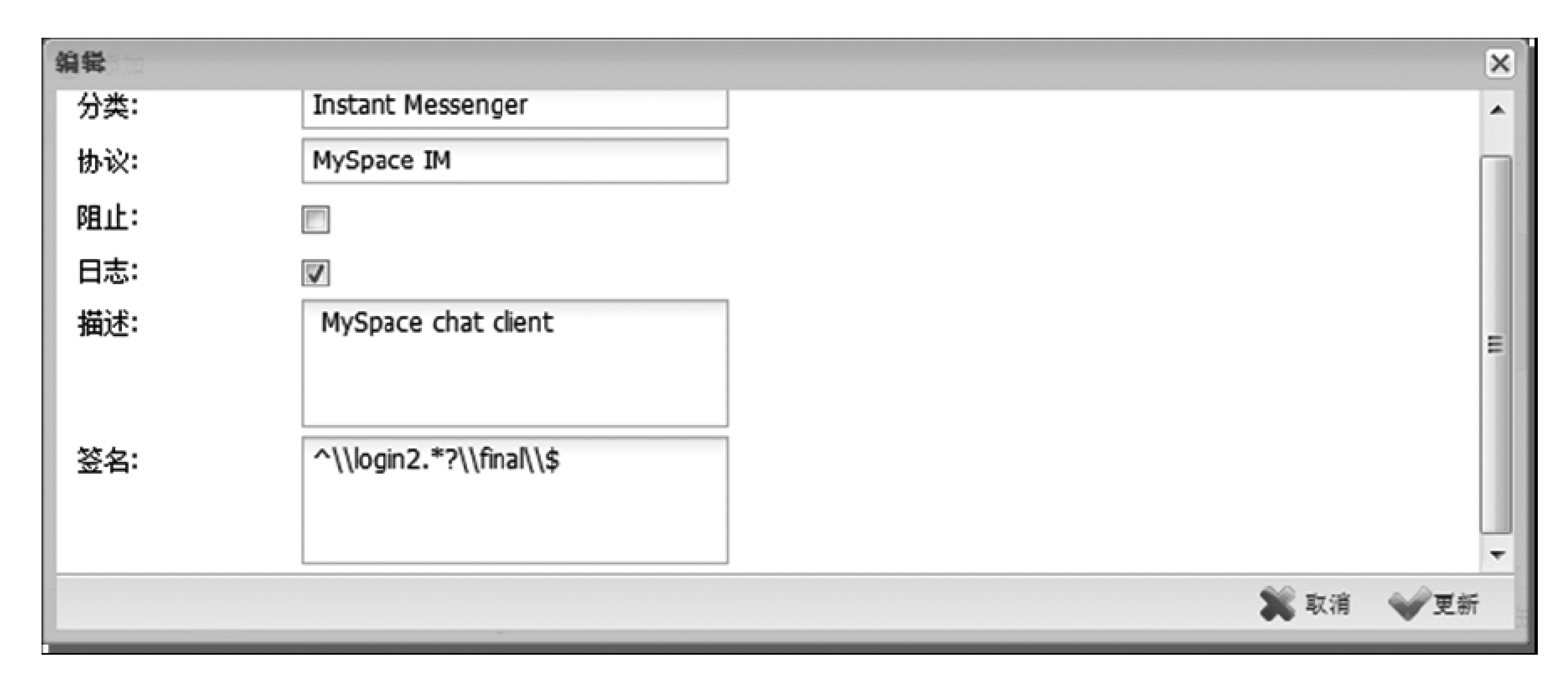


图 9-13 用户自定义协议控制

3. 事件日志

协议控制事件日志包括以下内容。

（1）时间戳：事件发生的时间。

（2）动作：是该流量采取的动作，阻断或者记录。

（3）客户端：该流量的客户端的 IP 地址。

（4）请求：请求的内容。

（5）动作原因：部署到该流量的规则。

（6）服务端：该流量的服务端的 IP 地址。

协议控制事件日志的界面如图 9-14 所示。



图 9-14　协议控制事件日志

9.5　服务质量

9.5.1　功能描述

服务质量(QoS)是带宽和时延的折中，一般来说越多的带宽利用率，延时就越高，而低带宽利用率，则有较低的时延。本模块通过为流量设置不同的优先级，将带宽和时延控制在合理范围内。

9.5.2　具体配置

1. QoS 策略

用户可以通过以下内容为流量配置 QoS 策略。

（1）端口。

（2）协议。

（3）IP 地址。

流量的优先级越高，系统处理该流就越迅速。本系统能够有效实现以下几点。

（1）为关键服务(如 VoIP)提供足够的带宽。

（2）降低时间敏感业务的时延，如 VoIP、SSH 等，以提高用户体验。

（3）在多个用户之间带宽公平分配，防止被少数用户抢占所有带宽。

（4）使用户充分使用有限的带宽，而不必通过升级网络带宽来改善网络状况。

（5）当网络中发生拥塞，本模块可以通过配置，保证关键业务带宽需求。

2. 默认规则

本模块提供多种默认的规则,分为以下两种。

(1) 不可编辑:Ping、TCP ACK 和游戏。用户不能编辑这些默认规则,因为它们包含大量的子规则,用户难以正确配置,所以系统预先自动配置这些规则,但用户可以设置这些规则的优先级。

(2) 可编辑:VoIP、DNS、SSH 等。用户也可以自行编辑默认的规则。

3. 不可编辑规则

(1) 在导航栏中选择配置→QoS。

(2) 指定不可编辑规则的速度和优先级。

QoS 配置如图 9-15 所示。

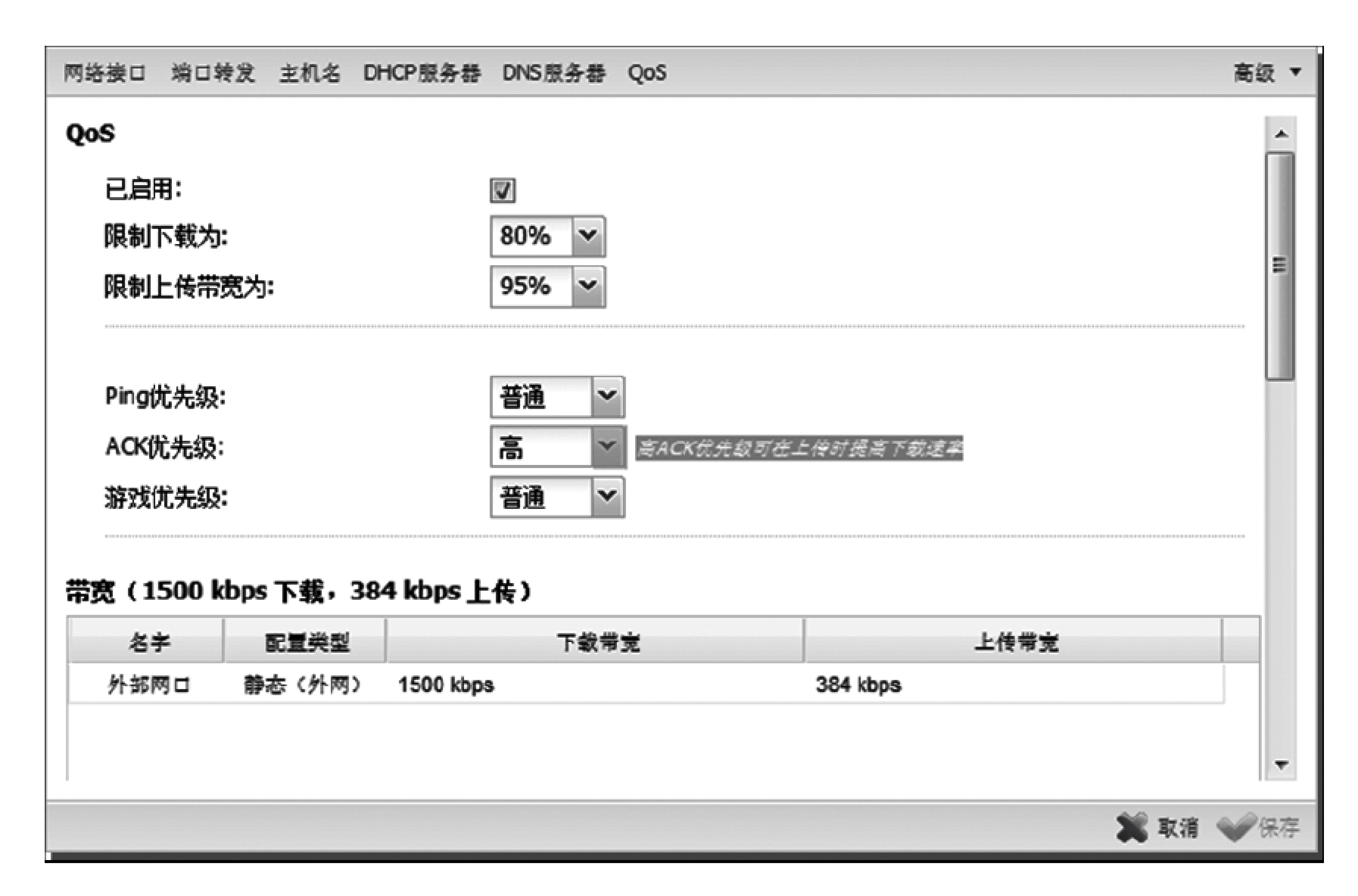


图 9-15 QoS 配置

QoS 中的字段名称及描述如表 9-3 所示。

表 9-3 QoS 中的字段名称及描述

QoS 中的字段名称	描　述
已启用	启用或者停止 QoS 功能
下载带宽	用户从 ISP 的最大下载带宽
限制下载为	实际下载带宽占最大下载带宽的比例,推荐配置为 80%~90%
上传带宽	用户上传的最大带宽
限制上传带宽为	实际上传带宽宽与最大上传带宽的比例,推荐配置为 90%~95%
Ping 优先级	ICMP 请求协议的优先级。默认配置为普通
ACK 优先级	ACK 确认的优先级,其与下载速度密切相关,提高其优先级能够提高下载速度和 VoIP 等协议的响应速度。默认配置为高
游戏优先级	默认规则包含 PS3、Wii、XBoX Live、Microsoft DirectX gameing 等协议,默认配置为普通

4. 可编辑规则

用户通过选择“开”复选框，激活默认可编辑规则，如图 9-16 所示。



图 9-16　QoS 规则编辑界面

单击“增加”按钮，添加或者编辑规则。

对于某条规则，用户可以从以下条件设置，如表 9-4 所示。

表 9-4　名称及描述

名　称	描　述
目的地址	流量的目的地址，如果不填，表示匹配所有的流量
目的为本地	流的目的为本机
目的端口	流的目的端口
协议	流的网络协议，此处只简单地设置了 7 种常见协议
源地址	流量的源地址，如果不填，表示匹配所有的流量
源接口	系统接收流的网络接口，有效值为外口、内控和 DMZ 网络接口
源端口	流的源端口

还可以设置一条新规则，参见图 9-17。

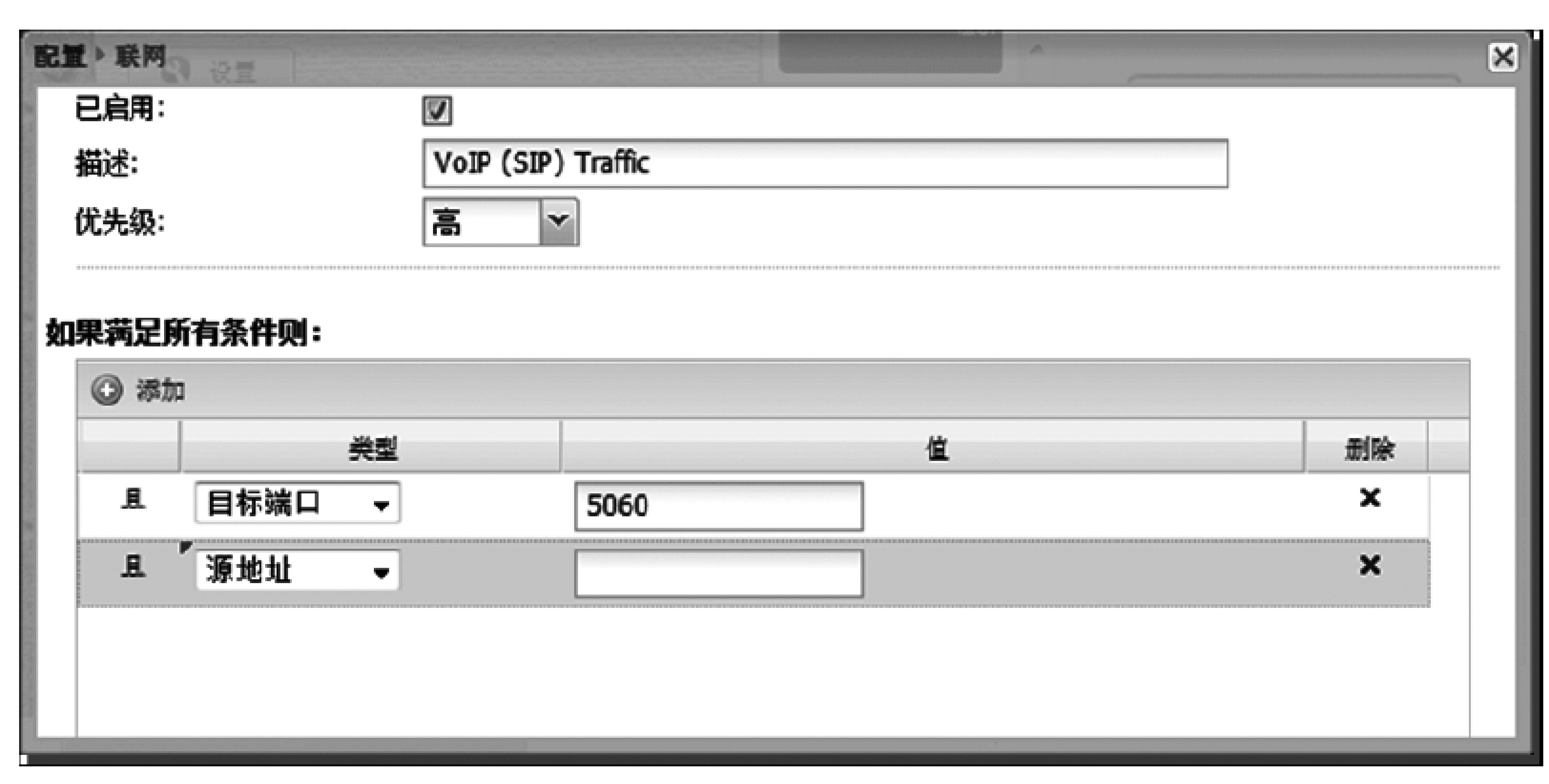


图 9-17　设置一条新规则

9.6　虚拟专用网

9.6.1　功能描述

豪讯虚拟专用网 Untangle VPN 支持 Site-to-Site 和 Client-to-Site 配置。当创建一个 VPN site 和客户端时，VPN 为每个用户创建一个定制可执行的 OpenVPN 客户端安装程序，客户端安装程序中包含该客户端认证和配置信息。用户只需要简单安装客户端安装程序就可以了。客户端安装程序支持 Windows Vista/Windows 7、Linux 和 UNIX 等操作系统。

一个虚拟专用网(VPN)是一个远程主机或网络和本地网络之间的通过不安全通路上(如互联网)建立的安全连接。VPN 连接包括客户端(Client)和服务端(Server)。Untangle UTM 配置为 VPN 服务器，允许远程客户端或站点连接到内部网络资源。

1. VPN 服务器和远程 VPN 客户端(软件)

当远程计算机客户端连接到 VPN 服务器时，软件 VPN 客户端和服务器建立了一条加密通信信道。每个 VPN 客户端通过唯一的安全密钥向服务器认证身份，这样服务器就可以保护网络资源免受不信任网络的访问。VPN 连接允许远程的客户端驻留在不安全的网络中，然而依然可以访问 VPN 服务器之后的受保护的资源，如图 9-18 所示。

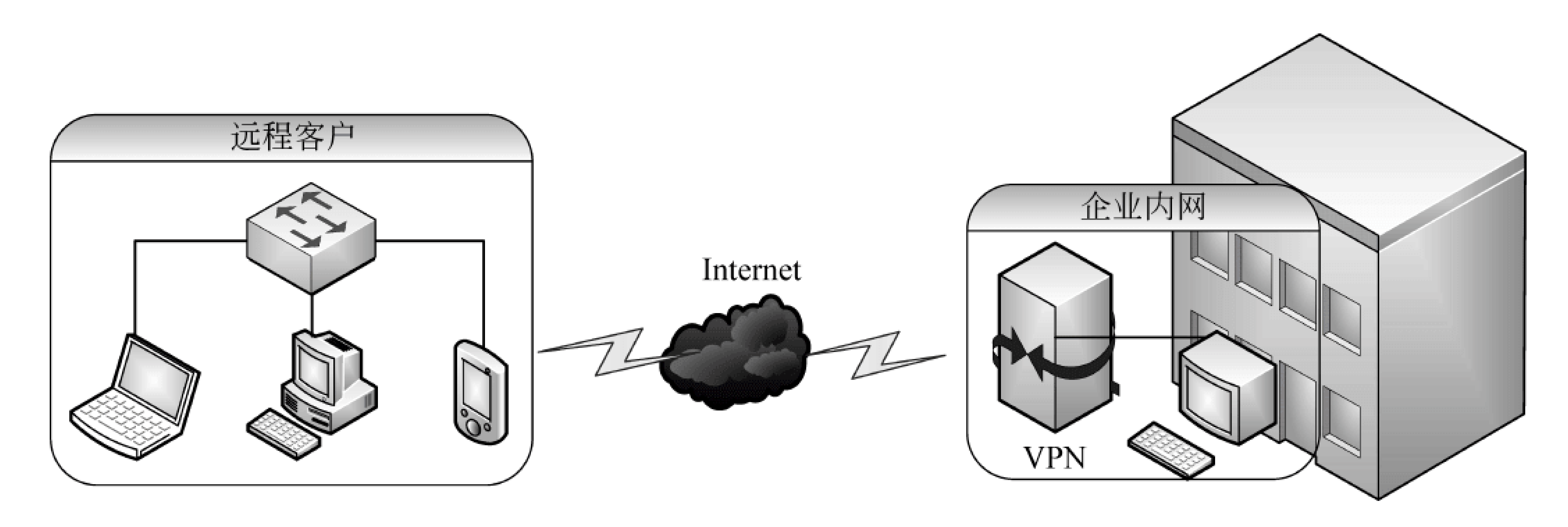


图 9-18　UTM 将 VPN 作为网桥工作模式示意图

2. VPN 服务器和远程 VPN 站点(硬件)

当整个远程网络连接到 VPN 服务器时，一个 VPN 站点代表了在受保护网络中许多单独的主机。这种配置是最常见的，如远程办公，其中很多员工需要加入公司总部的受保护网络中。当一组计算机或网络与服务器建立了 VPN 连接时，这组计算机就称为一个站点。Untangle UTM 可以作为一个远程站点，与另外一个远程位置 Untangle UTM 建立桥接，如图 9-19 所示。

9.6.2　具体配置

1. 密钥分发

每一个要和 VPN 服务器建立连接的计算机必须要安装密钥，这就是密钥分发。Untangle 服务器可以分发密钥通过 E-mail 或者 USB 盘的方式。

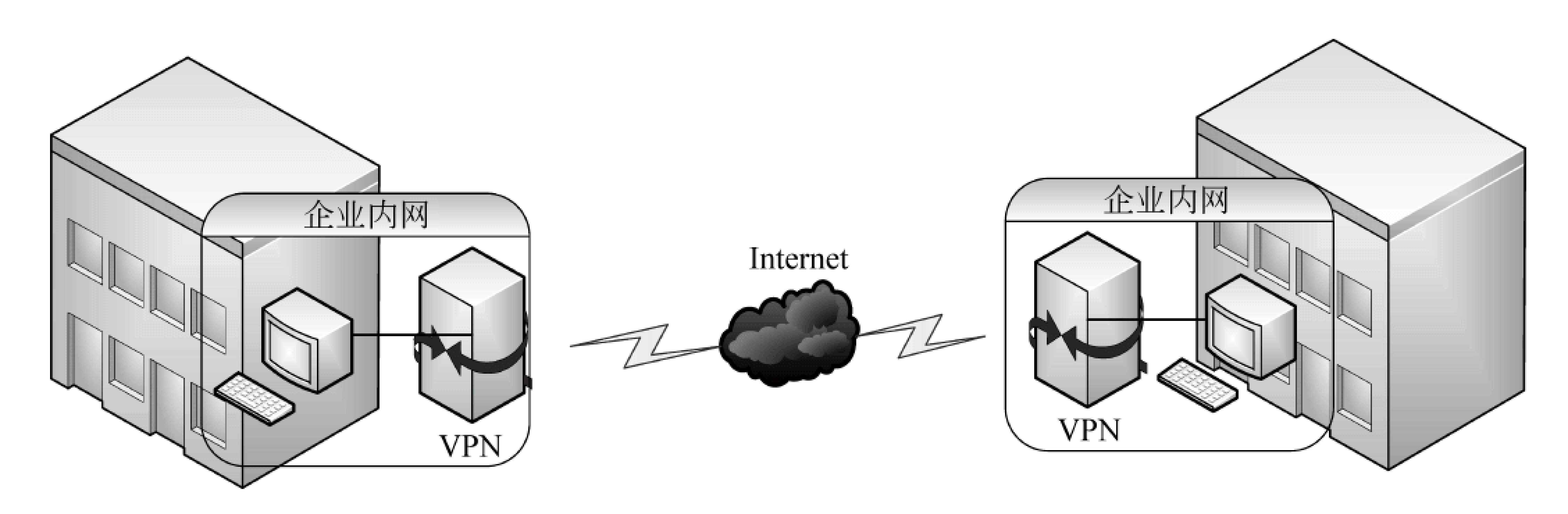


图 9-19 Untangle UTM 作为 VPN 站点互联示意图

2. 地址池

地址池是指 Untangle VPN 分配给 VPN 客户端的一段网络地址范围。这些 IP 地址是用于受保护的内部网络。例如，用户甲建立一个 VPN 连接到 Untangle VPN 服务器，他本来的 IP 地址为 166. *. *. *。当他和 VPN 服务器建立通信以后，他又获得了 VPN 服务器给他分发的 IP 地址 192.1.1.3。

3. 创建 VPN 的步骤

创建 VPN 的步骤如图 9-20 所示。

步　骤
(1) 进行必要的安装前准备工作，学习配置需求
(2) 配置Untangle UTM 为VPN服务器
(3) 添加VPN 站点和VPN客户端
(4) 分发密钥和VPN客户端
(5) 配置远程站点
(6) 访问网络资源，测试创建的VPN

图 9-20 VPN 配置步骤

1）配置 VPN 服务器之前的步骤

在试图连接分公司到总公司，或者提供客户端访问虚拟专用网络时，终端和地址池必须是相同的 IP 地址机制。如果终端主机不满足该条件，需要重新修改机器的 DHCP 服务器或者任何一个静态 IP 地址。

例如，地址池为 172.16.16.0，VPN 服务器的内部 IP 地址为 192.168.2.1，VPN 站点的内部 IP 地址是 10.0.0.1。则 VPN 服务器配置示意图如图 9-21 所示。

2）配置 Untangle UTM 为 VPN 服务器

设置虚拟专用网的第一步是配置 VPN 服务器。该过程使用 OpenVPN 设置向导来配置 Untangle UTM 为 VPN 服务器，这样的话 VPN 客户端和 VPN 站点可以连接到用户的受保护网络中。

（1）单击配置为 VPN 服务器，启动 VPN 设置向导，这时 VPN 欢迎页面出现，如图 9-22 所示。

（2）单击“下一步”按钮，出现证书页面。

（3）指定公司和位置信息，如图 9-23 所示。单击“下一步”按钮，系统将弹出输出页面。



图 9-21　VPN 服务器配置示意图

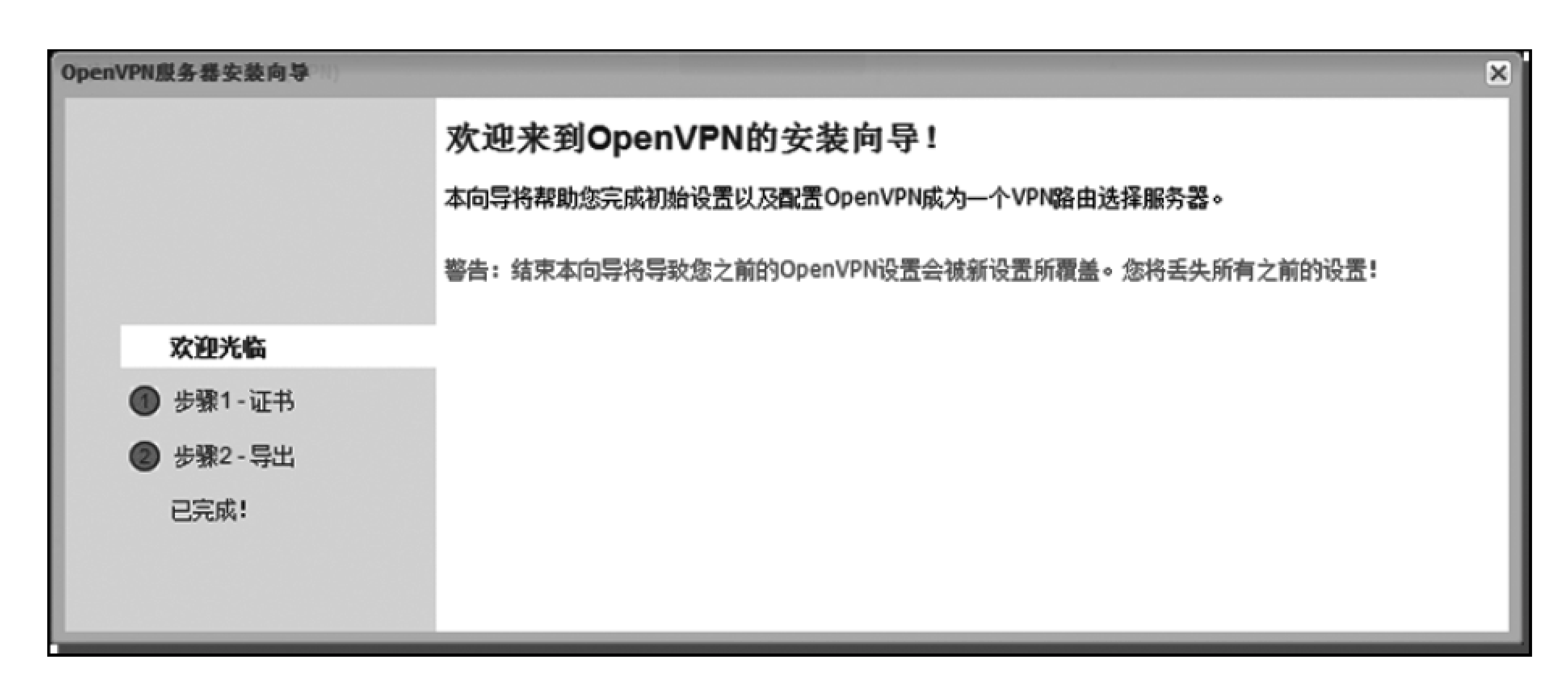


图 9-22　VPN 配置向导

OpenVPN服务器安装向导

步骤1 - 证书

请指定有关您位置的信息。该信息将用于生成一个安全的数字证书。

该信息是必须的。

组织: Hosun

国家: China

省: Beijing

城市: Beijing

欢迎光临

① 步骤1 - 证书

② 步骤2 - 导出

已完成!

图 9-23　VPN 配置向导步骤 1

(4) 加入另外的输出,编辑默认输出,或接受默认。默认情况下,VPN 输出整个网络。

(5) 如果你允许接受默认设置使得 VPN 用户可以访问整个网络,在主机/网络名称中填写资源的描述名称。

(6) 如果你仅允许 VPN 用户可以访问单台机器,在 IP 地址文本框中和 netmask 文本框中说明计算机的 IP 地址和 255.255.255.255 的子网掩码,然后在主机/网络文本框中填写资源的描述名称,如图 9-24 所示。



图 9-24　VPN 配置向导导出配置

(7) 单击“下一步”按钮。

(8) 单击“关闭”按钮来完成设置,如图 9-25 所示。这样 Untangle UTM 就可以配置为 VPN 服务器了。



图 9-25　VPN 配置完成

3) 添加 VPN 站点和 VPN 客户端

配置完 VPN 服务器后,我们来配置 VPN 站点和 VPN 客户端。

(1) 在 VPN 中单击高级选项。

(2) 在默认地址池中加入至少一个地址池,单击“下一步”按钮。

(3) 进行下一步操作。

(4) 配置 VPN 客户端，提供 VPN 客户端访问你创建的地址池，如图 9-26 所示。



图 9-26　配置 VPN 客户端

(5) 配置 VPN 站点，如图 9-27 所示，提供 VPN 站点访问你创建的地址池。



图 9-27　配置 VPN 站点

4) 分发密钥和 VPN 客户端

为了使 VPN 客户端和 VPN 站点能够访问 VPN 服务器，需要站点密钥。VPN 客户端也需要一个客户端。

生成密钥和分发 VPN 客户端的步骤如下。

(1) 在 VPN 中单击 VPN 客户端选项。

(2) 滑动到 VPN 客户端位置或 VPN 站点位置。

(3) 在分发列中，单击分发客户端，这时出现了分发客户端窗口，如图 9-28 所示。

(4) 在 VPN 客户端，指定用户的电子邮件地址和发送电子邮件按钮。Untangle UTM 发送电子邮件给用户一个连接以下载密钥和客户端。

5) 配置 Untangle UTM 为远程 VPN 站点

撤销用户的暂时 VPN 访问步骤如下。

(1) 在 VPN 中单击显示设置按钮。

(2) 单击 VPN 客户端选项。

(3)在 VPN 客户端区域中，清除 enabled 选项框，并单击“保存”按钮。

撤销用户的永久 VPN 访问步骤如下。

(1) 在 VPN 中单击显示设置按钮。

(2) 单击 VPN 客户端选项，删除对应用户账户的行。

(3) 生成新的客户端和密钥，转分发密钥和 VPN 客户端。

VPN 客户端的事件日志如表 9-5 所示。

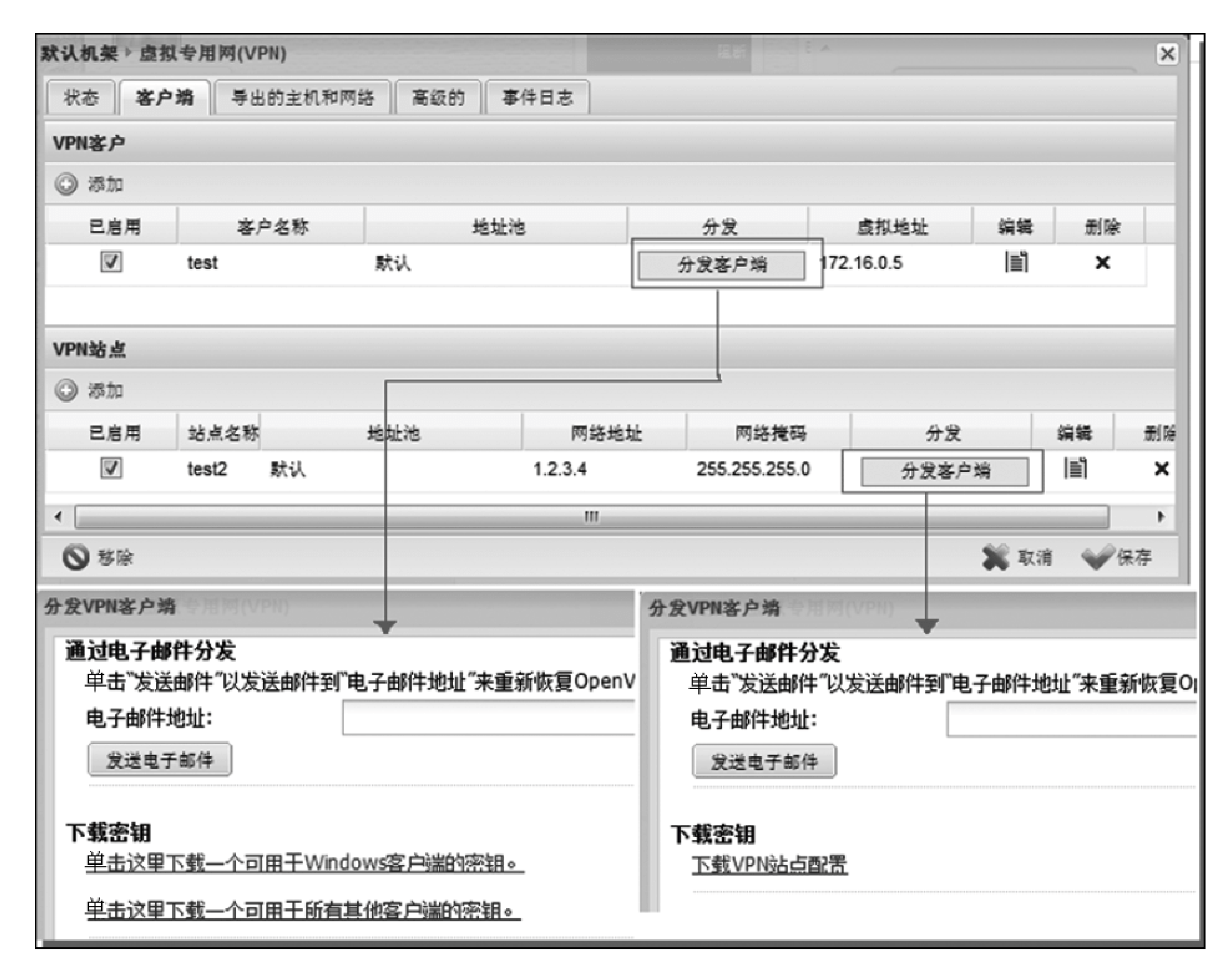


图 9-28　分发客户端窗口

表 9-5　VPN 客户端的事件日志

名　称	解　释	名　称	解　释
启动时间	VPN 连接的建立时间	客户端地址	VPN 连接客户端的 IP 地址
终止时间	VPN 连接的终止时间	发送的字节数	连接发送的字节数(KB)
客户端名字	VPN 客户端的名字	接收的字节数	连接接收的字节数(KB)

9.7　网页过滤

9.7.1　功能描述

网页过滤模块的主要功能是通过过滤含有不适当内容的网络流量的方式来保护网络。不同于其他软件，这一功能所采取的过滤标准是根据用户主观设置的。网页过滤模块过滤所依据的参数主要是分类数据、URL 地址、MIME 类型、文件类型。网页过滤面板如图 9-29 所示。



图 9-29　网页过滤面板

9.7.2 配置说明

1. 拦截指定网址的网络内容

默认的拦截列表可以将根据用户主观分类设置出来的不受欢迎内容大部分拦截。同样可以选择增加指定的网址，使用过程如下。

（1）从网页过滤模块中，单击“拦截列表”表项。

（2）单击“编辑网址”按钮。

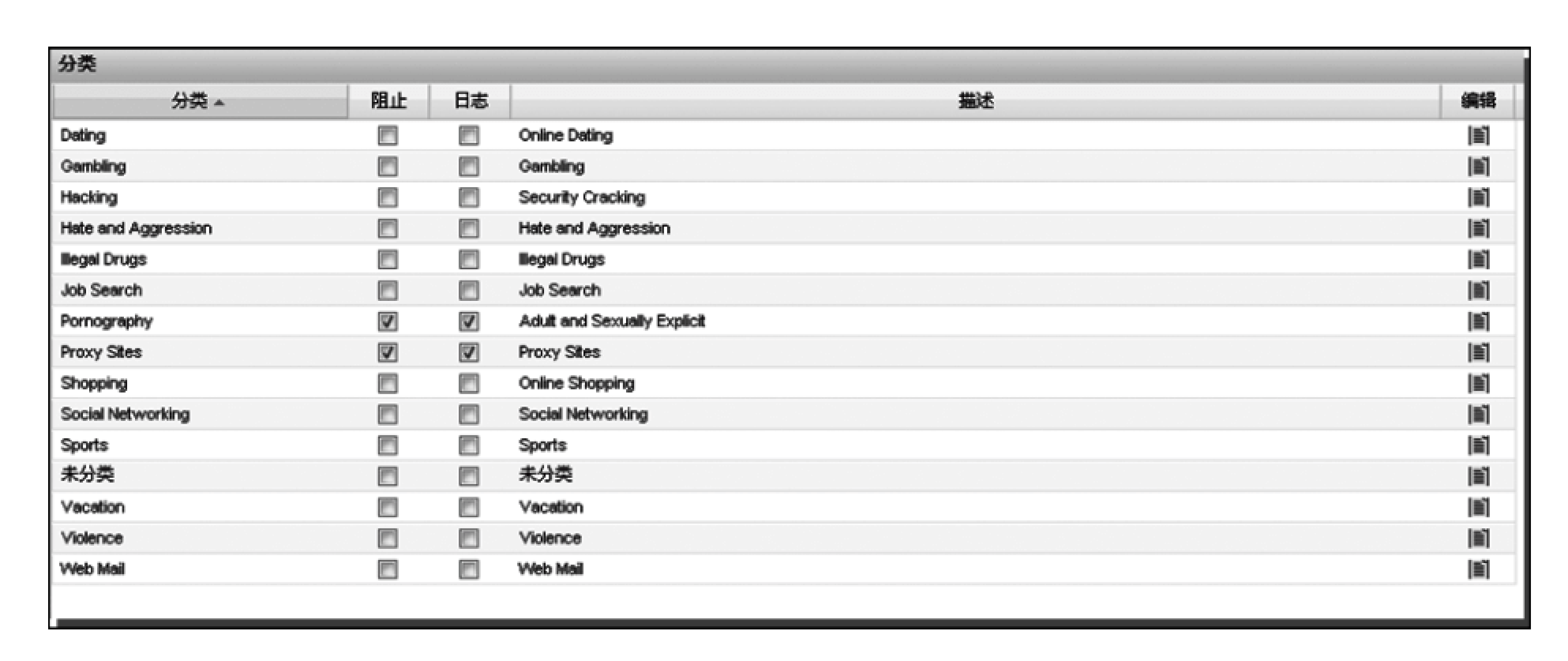
（3）在表格左侧，单击“增加”按钮。

（4）在新的条目中，增加你所需要指定的 URL 地址。

（5）单击“保存”按钮。

注意：如果你暂时不需要拦截该网址内容，可以不选择旁边的复选框。

分类拦截页面和网址拦截页面如图 9-30 和图 9-31 所示。



分类

分类	阻止	日志	描述	编辑
Dating	☐	☐	Online Dating	
Gambling	☐	☐	Gambling	
Hacking	☐	☐	Security Cracking	
Hate and Aggression	☐	☐	Hate and Aggression	
Illegal Drugs	☐	☐	Illegal Drugs	
Job Search	☐	☐	Job Search	
Pornography	☑	☑	Adult and Sexually Explicit	
Proxy Sites	☑	☑	Proxy Sites	
Shopping	☐	☐	Online Shopping	
Social Networking	☐	☐	Social Networking	
Sports	☐	☐	Sports	
未分类	☐	☐	未分类	
Vacation	☐	☐	Vacation	
Violence	☐	☐	Violence	
Web Mail	☐	☐	Web Mail	

图 9-30　分类拦截页面



默认机架 ‣ 网页过滤 ‣ 站点

站点

添加

站点	阻止	日志	描述
245.202.0.1	☑	☑	【无描述】

图 9-31　网址拦截页面

2. 放行指定网址的网络内容

如果贵公司认为某一指定网址非常有用，但同时因为内容和分类的关系正在被拦截中，或者你不再想要继续拦截之前设置的一个网址，你可以通过以下设置方式完成。

1）通过分类指定的 URL 拦截放行

（1）在网页过滤模块中单击“放行列表”表项。

（2）在网址区域中单击“管理列表”按钮。

（3）如果你想放行的 URL 出现在表格中，选择该 URL 对应的放行复选框；否则，如果你需要加入一条新的 URL，单击表格左侧的增加按钮，并在新的条目中增加你所需要放行的 URL 地址。

（4）单击“保存”按钮。

2）用户定义的 URL 地址放行

（1）在网页过滤模块中，单击“拦截列表”表项。

（2）在网址区域中，单击“管理列表”按钮。

（3）在表格中，选择你想放行的已经存在的 URL 地址，如图 9-32 所示，取消“拦截”复选框，或者简单地删除该行条目。

（4）单击“保存”按钮。

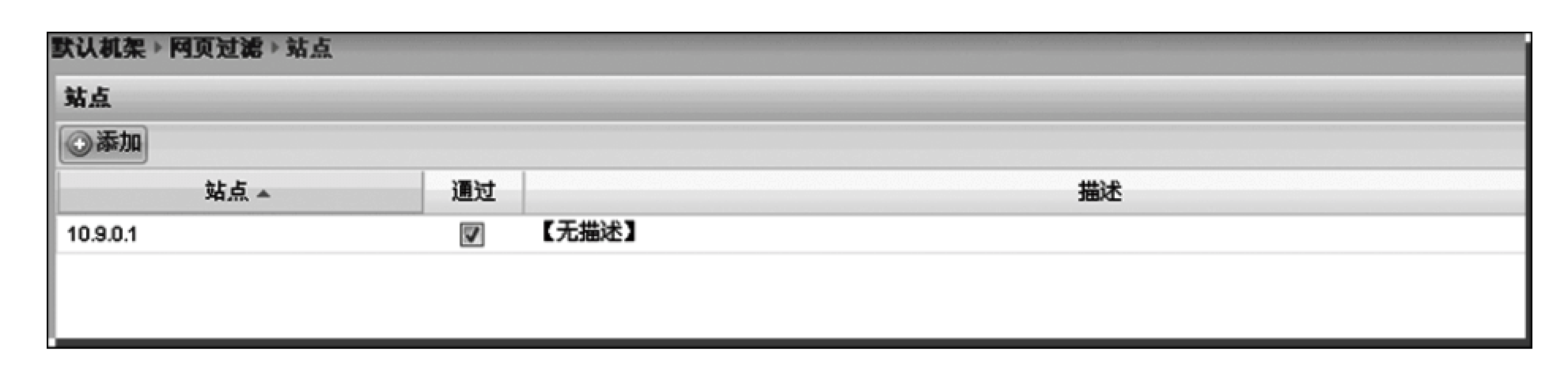


图 9-32　网址放行列表

3. 放行指定用户相关的网络内容

（1）从网页过滤模块中，单击“拦截列表”表项；从网址区域中，单击“管理列表”按钮。

（2）在表格中单击“添加”按钮，出现一条新条目，如图 9-33 所示。

（3）在 IP 地址/范围文本框中，指定你想从过滤名单中免除的用户计算机的 IP 地址和子网掩码。

（4）单击“保存”按钮。

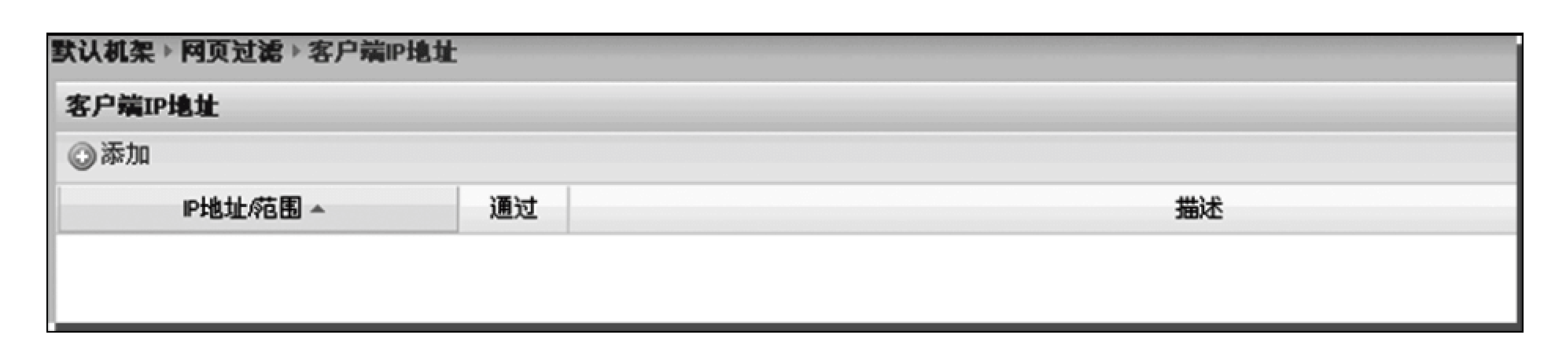


图 9-33　指定用户（地址）放行列表

4. 允许用户旁路指定网页

一些单位组织可能希望允许某些用户绕过网页过滤，这一选项在用户旁路。如果用户旁路栏设置为无，表示没有用户旁路已拦截的页面。但如果用户旁路栏设置为临时，用户将被允许旁路。如果用户旁路栏设置为永久和全局，那么用户被允许旁路的拦截网页将被加入到永久全局列表里。

用户旁路最好结合策略管理一同设置，这样只有某一些可以被允许旁路。设置如下。

（1）在网络过滤模块中单击“拦截列表”项。

（2）在用户旁路栏选择无、临时的或者永久和全局的，如图 9-34 所示。

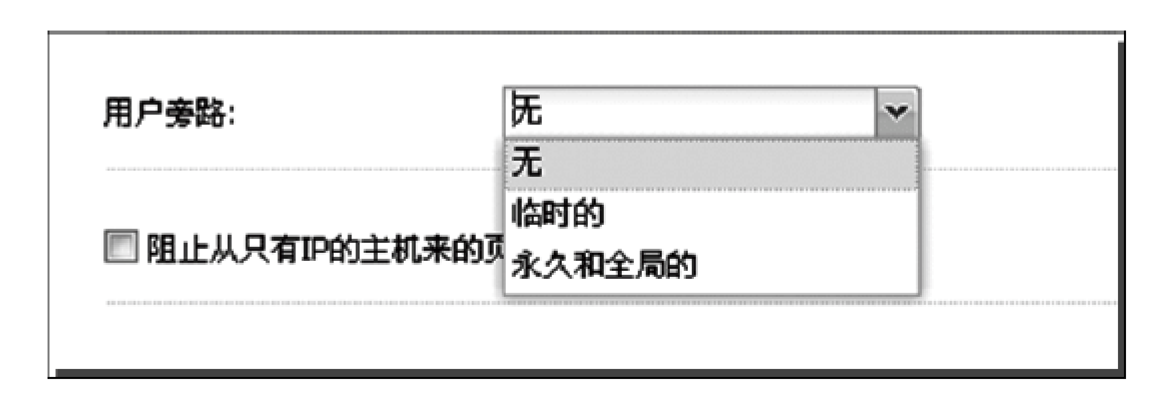


图 9-34　用户旁路选择界面

（3）单击“保存”按钮。

5．依据 MIME 类型来拦截网页内容

（1）在网络过滤模块中，单击“拦截列表”选项卡，再单击“MIME 类型”选项卡，如图 9-35 所示。

（2）如果你想拦截的 MIME 类型出现在表格中，则选择该 MIME 类型的拦截复选框。

（3）如果你需要添加一个新的 MIME 类型，单击表格左侧的“添加”按钮，并在新出现的条目中添加你所需要拦截的 MIME 类型。

（4）单击“保存”按钮。

默认机架 ▸ 网页过滤 ▸ MIME类型

MIME类型

添加

MIME类型 ▲	阻止	日志	描述
application/applefile	☐	☐	Macintosh File
application/dos-exe	☐	☐	DOS executable
application/exe	☐	☐	executable
application/futuresplash	☐	☐	Macromedia FutureSplash
application/mac-binhex40	☐	☐	Macintosh BinHex
application/mac-compactpro	☐	☐	Macintosh Compact Pro
application/macwriteii	☐	☐	MacWrite Document
application/msdos-windows	☐	☐	MS-DOS executable

图 9-35　MIME 类型拦截网页内容

6．依据文件类型来拦截网页内容

文件类型拦截网页内容如图 9-36 所示。

默认机架 ▸ 网页过滤 ▸ 文件类型

文件类型

添加

添加新行

件类型 ▲	阻止	日志	描述
avi	☐	☐	video
bin	☐	☐	executable
cab	☐	☐	executable
class	☐	☐	java
com	☐	☐	executable
cpt	☐	☐	compression
dll	☐	☐	executable
exe	☐	☐	executable
gif	☐	☐	image
hqx	☐	☐	archive

图 9-36　文件类型拦截网页内容

（1）在网页过滤模块中，单击“管理列表”按钮。

（2）如果你想拦截的文件类型已经出现在表格中，则选择该文件类型的拦截复选框或日志复选框或两者都选。

（3）如果你需要拦截的文件类型不在列表中，单击表格左侧的“添加”按钮，并在新出现的条目中，指定你需要拦截的文件类型。

（4）单击“保存”按钮。

9.8 攻击拦截

9.8.1 功能描述

攻击拦截模块的功能是用于以下几种方式来保护网络。

(1) 对所有 Untangle UTM 服务器接收到的数据包过滤查毒。这种包过滤清理功能是一个内嵌固定功能,不需要管理员设置。

(2) 保护网络免遭底层的网络攻击。

(3) 保护网络免遭 DoS 攻击。

攻击拦截页面如图 9-37 所示。

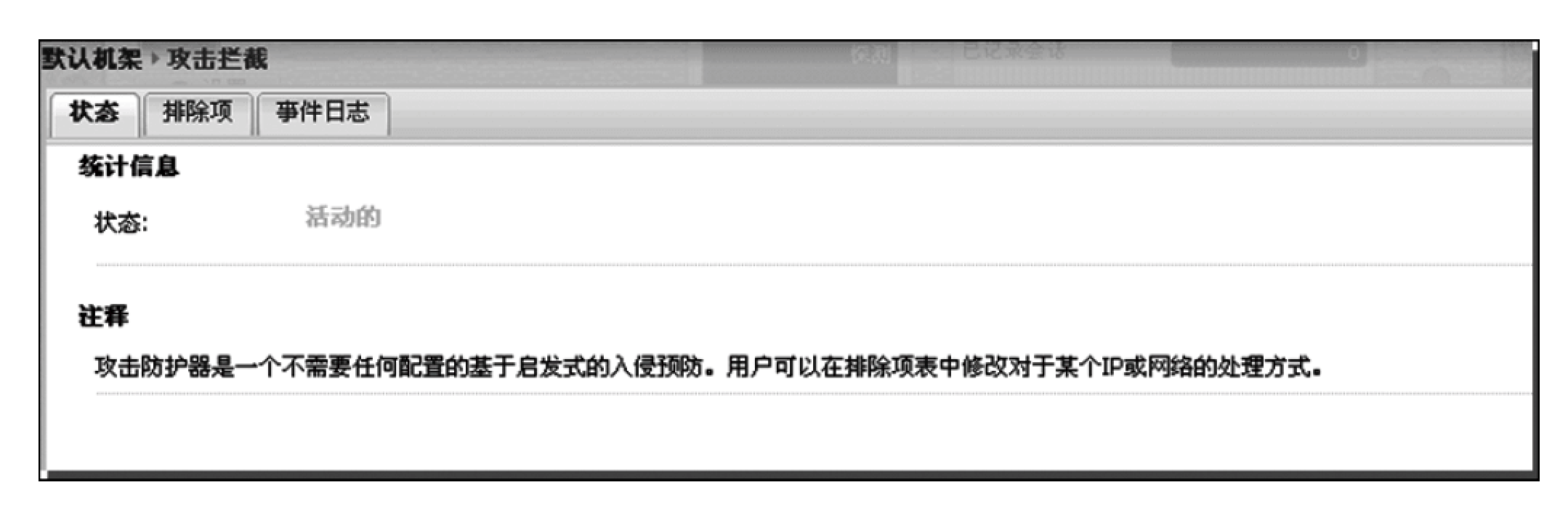


图 9-37　攻击拦截页面

9.8.2 抵抗拒绝服务攻击

攻击拦截需要追踪所有用户(IP 地址)的流量信息,同时连接数和数据容量也要被监控。如果一个特定用户很明显比其他用户更活跃,他的声誉值会增加。声誉值表示为一个在对应范围内的数字。大的声誉值表示该用户在消耗比其他节点更多的网络资源(更多连接数,更多传输的字节数)。

随着 Untangle UTM 服务器的负载增加,可能没有足够的系统资源来服务所有请求。不同于降低每一个请求的服务速率,攻击拦截模块只会对最高声誉值的用户采取行动。通过这种方式,占用所有带宽的用户将被分配更少的资源,同时那些需求量少的用户将保持固定服务性能级别不变。针对大声誉值的用户攻击拦截模块总共有三类可选行为。

9.8.3 攻击拦截行为

(1) 限制:攻击拦截模块会限制用户访问受保护的网络内部资源,使用这一限制的用户会感觉到网络性能会略微下降。

(2) 丢弃:攻击拦截模块会丢弃某一用户的数据包,比限制行为降低更多的网络流量。

(3) 拒绝:攻击拦截模块会拒绝某一用户的某一个指定会话的数据包。

通过使用声誉值的方式来分配资源,攻击拦截模块保护了整个网络抵御拒绝服务攻击。当一个用户试图针对 UTM 服务器保护的网络发起洪泛攻击,攻击用户的声誉值将增加,他的网络性能将从受限制的降低,直到完全拒绝访问受保护的资源。

Untangle UTM 服务器所采取的行为必须依据用户的声誉值。拒绝用户的行为只有极端情况才会出现,攻击拦截模块会在允许用户活跃和关闭该用户会话之间有明显清晰的分界线。不管怎么用,攻击拦截模块觉得该用户的活跃程度达到威胁网络时,就会拒绝他的会话。大部分情况下,限制用户访问和丢弃用户数据包的行为就足够保护网络了。

攻击拦截模块没有什么包过滤杀毒的设置,但能够设置指定某一用户因声誉值高而明显不同于其他节点。这部分可以通过添加攻击拦截异常部分完成。攻击拦截日志页面如图 9-38 所示。

默认机架 ▸ 攻击拦截

状态 | 排除项 | 事件日志

时间戳	源	源接口	名声	受限制的	已抛弃	拒绝
2009-12-22 4:03:51 pm	166.111.137.193	eth1	73.866387	13	0	0
2009-12-22 4:03:41 pm	166.111.137.193	eth1	90.943891	14	1	0
2009-12-22 4:03:31 pm	166.111.137.193	eth1	96.504791	16	2	0
2009-12-22 4:03:11 pm	166.111.137.193	eth1	96.565111	4	5	0
2009-12-22 4:03:01 pm	166.111.137.193	eth1	70.068541	2	0	0
2009-12-22 4:02:11 pm	166.111.137.193	eth1	60.337077	1	0	0
2009-12-22 4:02:01 pm	166.111.137.193	eth1	73.372572	15	0	0
2009-12-22 4:01:51 pm	166.111.137.193	eth1	98.698915	20	7	0
2009-12-22 4:01:41 pm	166.111.137.193	eth1	101.335053	13	12	0
2009-12-22 4:01:31 pm	166.111.137.193	eth1	79.598019	12	0	0
2009-12-22 3:59:20 pm	166.111.137.193	eth1	61.088977	1	0	0

图 9-38 攻击拦截日志页面

9.8.4 添加攻击拦截排除项

使用该异常列表是用来标识一台虚拟计算机(IP 地址),从而标识多于一个的物理计算机的。如前面抵御拒绝服务攻击部分,攻击拦截模块会追踪其网络内相关的活跃计算机。如果某一个 IP 地址代表多于一台物理计算机,例如,采用了 NAT 方式,那么攻击拦截模块必须知道该 IP 地址,否则,拦截模块会认为是某单一计算机的异常活跃而导致拒绝整个网络的流量。具体的配置操作如下。

(1) 在攻击拦截模块中,单击"设置"按钮。

(2) 单击"排除项"选项卡,如图 9-39 所示。

(3) 单击"添加"按钮。

(4) 指定你需要添加的异常:开启(当该选框选中,标识异常规则开启),地址(需要被指定的计算机 IP 地址),用户数(该地址所表示的用户或计算机数量,例如,如果 5 个用户在 NAT 系统后面,那该 NAT 系统的外部 IP 地址就需要 5 个用户数)。

(5) 单击 OK 按钮。



图 9-39 添加排除项页面

9.9 广告拦截

9.9.1 功能描述

Untangle UTM 广告拦截组件可以拦截大多数通过网页推送给用户的广告。广告拦截组件使用可更新的过滤表,该表包括通常被用于推送广告的链接和扩展。

9.9.2 具体配置

当 Untangle UTM 安装后,它将下载配置文件。该配置文件由大量网站及网站扩展组成,它们均已被发现为用于推送广告。对于大多数用户来说,默认的配置已经足够。但你仍可以做调整,手动添加过滤表项和白名单。

1. 解除阻止某个网页

如果合法的网页内容被阻止了,你可以简单地解除阻止。单击“配置”进入广告拦截的设置页面,然后选择“事件日志”查找被阻止的相关页面。然后选择“过滤器”栏,找到对应的规则,将其修改或禁用,注意保存你的修改。

2. 阻止某个网页

如果希望添加一条新的阻止规则,单击“设置”进入广告拦截的设置页面,选择过滤器栏。单击左上角的“添加”按钮,添加新的过滤规则,并将其选为阻止。

3. 使用白名单列表

如果你访问的某个网站被多个不同的规则阻止,你可以将该网站加入白名单列表。单击“设置”按钮进入广告拦截设置页面,然后选择“通过列表”选项卡。单击管理列表来添加站点(如 google. com)和额外的说明。当然,也可以删除曾经添加过的站点。

9.10 防间谍软件

9.10.1 功能介绍

防间谍软件模块由多个开源项目组成。防间谍软件模块检查来自受保护网络中的 Web 请求,而后采取以下操作。

(1) 使用病毒特征码检测和识别特定病毒。

(2) 阻止键盘记录软件。

(3) 提供一个包含已知间谍软件网站的 URL 黑名单。

(4) 提供一个需要 Cookies 的网站的 URL 黑名单。

(5) 阻止已知的用于间谍软件的恶意 ActiveX 控件。

(6) 检查用户所访问网站的 IP 地址,并将它们与不受欢迎子网列表进行比较。

9.10.2 配置说明

1. 解除对间谍网站的阻止

如果在受保护网络中一个可信用户需要访问阻止名单上的某个站点,你可以解除对这

个网站的阻止；但是，为了保证工作场所更高的安全级别，Untangle UTM 不推荐这么做。

如果要防止某个网站被划为间谍网站，可以将该站点添加到“通过”列表，或者使用本节介绍的防间谍模块的“快速通过”列表。如果要指定某些而非全部用户拥有这些特权，可以为这些用户创建一个虚拟机架。

要解除对一个间谍网站的阻止，需进行如下操作。

(1) 在防间谍软件模块中，单击“设置”按钮。

(2) 采取如下步骤之一：

① 如果希望由用户来决定使本人或他人或者全部用户开启或关闭对某个已知间谍软件网站的阻止，可以指定一个“快速通过”列表。

a. 单击“阻止列表”选项卡。

b. 在用户旁路设置中，选择临时或永久和全局，如图 9-40 所示。



图 9-40 设置用户旁路有效期

临时：使用户能够访问间谍软件网站 1 个小时。当该用户访问间谍软件网站时，用户会收到一条告警信息通知用户该网站是已知的间谍软件网站。用户可以绕过该告警并访问站点，或者选择不登录该站点。如果用户绕过了告警，防间谍软件模块将在 1 小时内不会再向该用户发出告警。如果用户选择不登录该网站，下次该用户访问该站点时，防间谍软件将警示用户登录该站点的风险。

永久和全局：使所有用户能够访问间谍软件网站，并对自己或所有其他人解除对该站点的阻止。当用户访问一个间谍软件网站时，用户会收到一条告警信息通知用户该网站是已知的间谍网站。该用户可以绕过告警并访问该站点，或者选择不登录该站点。如果用户绕过了告警，防间谍软件模块将不再向所有用户发出告警。如果用户选择不登录该站点，防间谍软件将持续警告用户登录该间谍软件站点的风险。

② 如果希望能够放行特定站点，可以创建一个通过列表。

a. 单击“通过列表”选项卡。

b. 单击“添加”按钮，将会出现一个编辑框，其中“通过”复选框是选中的。

c. 在编辑栏中输入希望放行的站点域名。域名是唯一需要填的部分。域名的格式为 http://domain_name，但是只识别第一层域名。例如，如果在通过列表中输入 http://www.untangle.com/news，那么 http://www.untangle.com 下的所有路径都会被放行。

注意：管理员可以通过选择或不选择通过复选框来放行/阻止某个域名通过。

2. 取消对 ActiveX 控件的阻止

如图 9-41 所示，轻易不要解除对恶意 ActiveX 控件的阻止。防间谍软件模块为用户提供解除对 ActiveX 控件的阻止功能。然而，Untangle UTM 建议不要解除对任何默认 ActiveX 控件的阻止设定，因为这些都是已知的恶意 ActiveX 控件。如果在防间谍软件模块中添加了任何其他的 ActiveX 控件，你就可以使用“阻止”复选框来启动或禁用该 ActiveX 控件，以保证用户能够正确识别 ActiveX 控件。



图 9-41　默认 ActiveX 控件列表

1）要阻止其他 ActiveX 控件

(1) 在防间谍软件模块中，单击“设置”选项卡。

(2) 单击“阻止列表”选项卡，如图 9-42 所示，再单击“管理列表”按钮。出现的表格包含了已知的恶意 ActiveX 控件，因此请不要清空“阻止”复选框。

(3) 单击“添加”按钮，将会出现新的一行，其中“阻止”复选框是被勾选的。

(4) 添加 ActiveX 控件标识，然后单击“保存”按钮。

2）阻止所有 ActiveX 控件

虽然防间谍软件模块提供了一个已知恶意 ActiveX 控件列表，但这并不完善，因为并非



图 9-42　防间谍软件配置

所有恶意 ActiveX 控件都为人所知。要阻止所有 ActiveX 控件,进行如下操作。

(1) 在防间谍软件模块中,单击“设置”按钮。

(2) 勾选“阻止所有 ActiveX”复选框,见图 9-42。

(3) 单击“确定”按钮。

3. 防间谍模块的时间日志

(1) 时间戳:事件发生的时间。

(2) 行动:发生的行动(例如阻止)。

(3) 客户端:流量的客户端 IP 地址。

(4) 请求:有关请求的描述(例如 http://someurl/somepath.thml)。

(5) 行动原因:采取行动的原因(例如在 URL 列表中)。

(6) 服务器:流量的服务器 IP 地址。

9.11　钓鱼网站拦截

9.11.1　功能介绍

钓鱼网站拦截模块提供两个反钓鱼和一个反域名欺骗保护功能。

1. 电子邮件反钓鱼保护

钓鱼网站拦截可以检查钓鱼邮件、欺骗邮件。钓鱼邮件一般都会通过伪造信任用户邮件等欺诈手段来获取敏感信息,例如密码和信用卡信息等。

2. Web 反钓鱼保护

钓鱼网站拦截使用谷歌黑名单来进行反钓鱼保护,谷歌黑名单包含已知的欺骗网站。为了提高保护水平,Untangle UTM 服务器随谷歌黑名单一起每 6 小时更新一次钓鱼网站拦截。即使你禁用了 Untangle UTM 的自动跟新功能,钓鱼网站拦截也会接收谷歌黑名单

更新。

如果你单击了钓鱼邮件中的 URL 链接，该链接将转向某个非法站点，而该站点又在谷歌黑名单中，那么当你访问该站点时，系统将会告警，防止你进入该站点。

对 Web 钓鱼攻击进行过滤如图 9-43 所示。



图 9-43 对 Web 钓鱼攻击进行过滤

3. Web 仿冒保护

除了对钓鱼邮件和网站过滤外，Untangle UTM 的钓鱼网站拦截模块还会阻止仿冒站点。

某些站点模仿合法站点(通常是银行或电子商务平台)，并使用社会工程学的方法来骗取用户的敏感信息。这些仿冒站点通常伪造与合法站点很相似的域名，以此来欺骗用户。

9.11.2 配置说明

1. 配置钓鱼邮件扫描

(1) 在钓鱼网站拦截模块中，采取以下步骤，如图 9-44 所示。



图 9-44 选择协议配置

① 如果是本地微软 Exchange 服务器,则使用 SMTP。
② 如果通过 Outlook 来下载 Web 邮件,则使用 POP3。
③ 如果采用的是 IMAP 邮件客户端,则使用 IMAP。

(2) 如表 9-6 所示,选择你希望 UTM 采取的行动。

表 9-6　字段及描述

字　段	描　述
扫描 SMTP/POP3/IMAP	勾选该复选框后,UTM 服务器将对邮件进行双向扫描,除非自定义策略覆盖了这条指令
行动	控制 UTM 服务器采取什么行动。 标记:对邮件进行标记,是用户可以通过设置过滤规则来将这些邮件放到不同文件夹下。 通过:放行邮件,即使它被检测为钓鱼邮件。 阻止:仅对 SMTP 有效。阻止邮件,意味着发件人认为邮件已经被转发,实际上邮件无法到达收件人。虽然发件人和收件人并不知道邮件被阻止,但该操作会记入时间日志。 隔离:仅对 SMTP 有效。将邮件隔离。细节可参见“隔离”章节。正如“创建自定义策略”一节描述的,向外发送的邮件默认情况下不会被隔离

(3) 单击“保存”按钮。

2. 钓鱼网站拦截时间日志

1) Web 时间日志

(1) 时间戳:事件发生的时间。

(2) 行动:对邮件采取的操作。其值依赖于邮件协议,但都包含描述文本。

(3) 客户端:协议客户端 IP 地址。对 SMTP 来说是邮件发送者,对 IMPA/POP 来说是邮件接收者。

(4) 请求:关于请求的描述。

(5) 服务器:服务器 IP 地址。对 SMTP 来说是接收邮件的机器,对 IMPA/POP 来说是收件箱机器。

2) 邮件时间日志

(1) 时间戳:事件发生的时间。

(2) 行动:对邮件采取的操作。其值依赖于邮件协议,但都包含描述文本。

(3) 客户端:协议客户端 IP 地址。对 SMTP 来说是邮件发送者,对 IMPA/POP 来说是邮件接收者。

(4) 主题:邮件主题,可以为空。

(5) 接收者:邮件的接收地址。

(6) 发送者:邮件的发送人。对于钓鱼邮件来说,这一项总是非法的。

(7) 服务器:服务器 IP 地址。对 SMTP 来说是接收邮件的机器,对 IMPA/POP 来说是收件箱机器。

9.12 垃圾邮件拦截

9.12.1 功能介绍

垃圾邮件拦截是一个智能邮件过滤器，能够识别垃圾邮件，包括以图片形式发送的垃圾邮件。单击垃圾邮件拦截模块的“配置”按钮，进入配置界面，如图 9-45 所示。

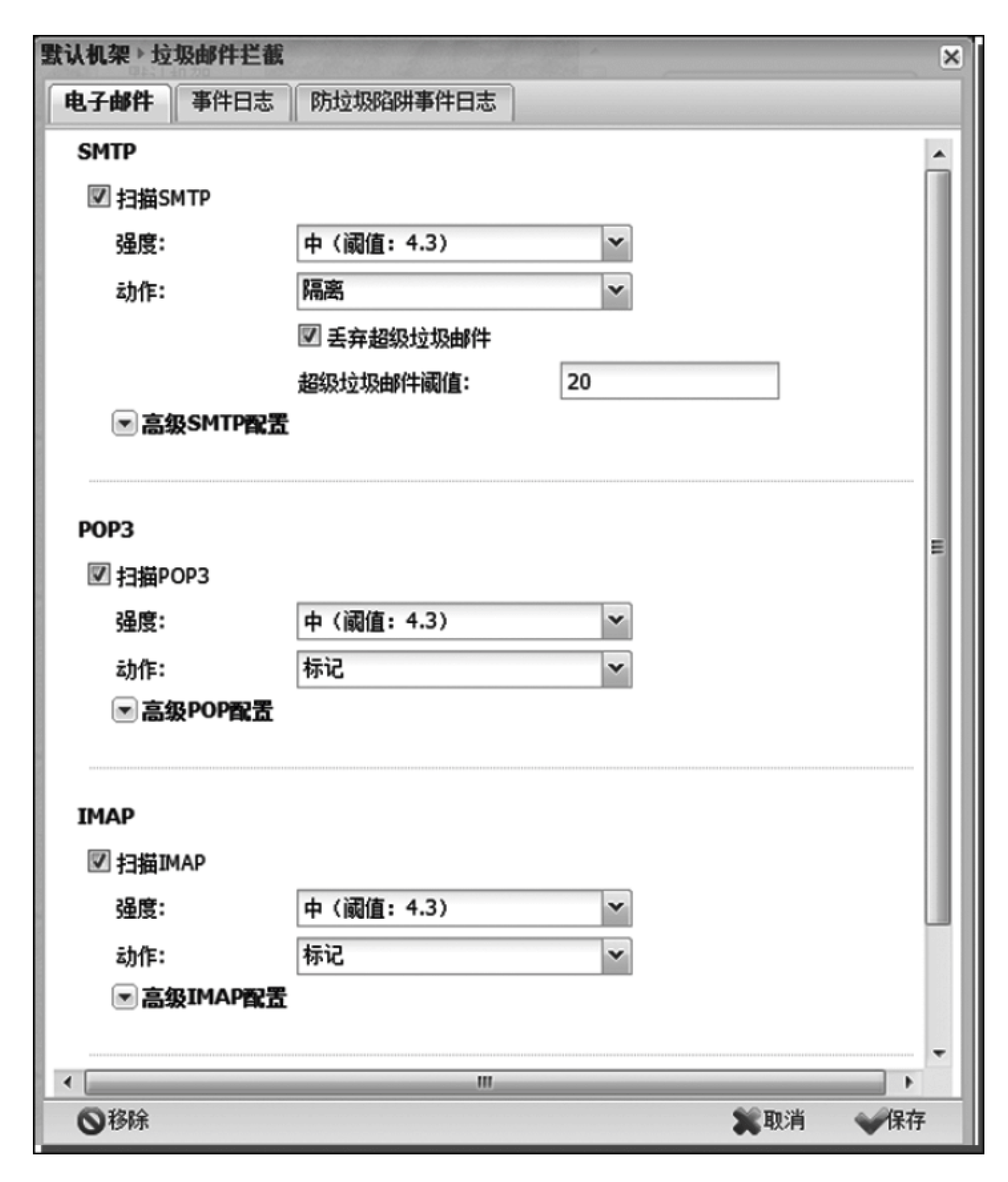


图 9-45 垃圾邮件拦截配置

1. 支持的协议

可以扫描通过下述协议传输的邮件：SMTP、POP、IMAP。可以自定义如下垃圾邮件拦截操作。

（1）扫描垃圾邮件。

（2）通知垃圾邮件用户。

（3）管理垃圾邮件。

2. 自定义

通过用户接口，可以定义阈值为严格、宽大或者介于两者之间的某个值。同时，可以为设置垃圾邮件拦截进行以下操作。

（1）标记：在邮件主题部分插入“垃圾”字样，并允许用户将垃圾邮件放入指定文件夹。

（2）通过：放行邮件，不对邮件进行处理。

(3) 丢弃：丢弃邮件，且不通知接收者，但会记入时间日志。

(4) 隔离：隔离邮件，供用户将来查看和处理。

3. 保守阈值

垃圾邮件拦截基于几百条特征码来识别垃圾邮件，每个特征码都有权重，可以生成一个综合分数。垃圾邮件拦截通过该综合分数来判断邮件是垃圾的概率。综合分数加上一个阈值(扫描力度，可以自定义)就可以确定邮件是否为垃圾。

默认情况下，垃圾邮件拦截阈值设置为中等。该阈值可以阻止大多数垃圾邮件而不会影响合法邮件。如果设置阈值在中等之上，垃圾邮件拦截就会更严格检查，从而可能会将一些合法邮件标志位垃圾。Untangle UTM 推荐使用“中等”阈值，因为 Untangle UTM 的目标是对合法邮件的零误判率。也就是说，Untangle UTM 不希望将非垃圾邮件错判为垃圾邮件。大多数用户偏好这种设置，但用户完全可以自定义。另外，垃圾邮件拦截会不断更新特征库。

9.12.2 配置邮件扫描和隔离

用户可以隔离所有 SMTP 邮件，或者指定垃圾邮件拦截隔离发给特定用户的垃圾邮件。对于 POP 和 IMAP 邮件，没有隔离选项，并且用户不能阻止这类邮件，因为必须将这些邮件下载下来才能查看，但可以标识它们为垃圾邮件。具体配置步骤如下。

(1) 在垃圾邮件拦截模块中，采取下述操作。

① 如果是本地微软 Exchange 服务器，则使用 SMTP。

② 如果通过 Outlook 来下载 web 邮件，则使用 POP3。

③ 如果采用的是 IMAP 邮件客户端，则使用 IMAP。

(2) 指定 UTM 的行为，如表 9-7 所示。

表 9-7 UTM 的行为

字　　段	描　　述
扫描 SMTP/POP3/IMAP	勾选该复选框后，UTM 服务器将对双向的邮件进行扫描，除非自定义策略覆盖了这条指令
力度	可以控制垃圾邮件拦截的灵敏度，共有高低 5 个值和一个自定义值。注意，最高阈值对垃圾邮件最为灵敏，会使大部分邮件被判定为垃圾邮件
动作	可以控制 UTM 对邮件采取的动作。 标记：在邮件主题上标上“垃圾”字样。用户可以设置邮件过滤规则，并将这些邮件放置到指定文件夹。 通过：即使检测为垃圾邮件，也会将其发送给接收者。 丢弃：仅适用于 SMTP 邮件。将邮件丢弃。发送者和接收者都不会知道邮件被丢弃，但会记入时间日志。 隔离：仅适用于 SMTP 邮件。将邮件隔离。在默认情况下，向外发送的邮件不会被隔离。注意，可以为超级垃圾邮件设置阈值，这样所有超过某个检测分数的垃圾邮件将直接被丢弃，而不是被隔离

可添加的高级配置如表 9-8 所示。

表 9-8　可添加的高级配置

字　　段	描　　述
添加邮件头部	勾选复选框后,UTM 服务器将在每封邮件头部添加信息,以说明该邮件是否为垃圾邮件、分数和做出该决定的检测信息
邮件大小限制	允许修改被检测邮件的最大尺寸。默认为 262 144B。垃圾邮件一般都比较小,但数量较多
启用“缓送”功能	仅适用于 SMTP 邮件。如果勾选了该项,将启用 DNSBL 功能,阻止黑名单上的邮件主机的连接
扫描失败时关闭连接	仅适用于 SMTP 邮件。如果垃圾邮件拦截失效,该设置可以决定进来的邮件是否可以不用检测,或者将其阻止直到检测完成
扫描外连邮件(WAN) SMTP	仅适用于 SMTP 邮件。该设置决定垃圾邮件拦截是否检测向外发送的邮件
CPU 负载限制	仅适用于 SMTP 邮件。如果 CPU 负载超过某个值,向内的连接将会被停止,直到 CPU 负载减低。默认值是 7
当前扫描限制	仅适用于 SMTP 邮件。同时扫描的最大邮件数。默认值是 15

9.12.3　关于垃圾邮件拦截时间日志

单击“事件日志”选项卡,显示如图 9-46 所示界面,其中各字段含义如下。

(1) 时间戳:事件发生的时间。

(2) 动作:对邮件采取的操作。其值依赖于邮件协议,但都包含描述文本,如阻止或标记。

(3) 客户端:协议客户端 IP 地址。对 SMTP 来说是邮件发送者,对 IMPA/POP 来说是邮件接收者。

(4) 主题:邮件主题。可以为空。

(5) 接收人:邮件的接收地址。

(6) 发送人:邮件的发送人。对于垃圾邮件来说,通常是空的。

(7) 垃圾邮件分数:垃圾邮件扫描时使用的分数。高分说明更有可能是垃圾。

(8) 服务器:服务器 IP 地址。对 SMTP 来说是接收邮件的机器,对 IMPA/POP 来说是收件箱机器。



图 9-46　垃圾邮件拦截事件日志显示界面

9.13 防　　控

9.13.1　功能介绍

Untangle UTM 为用户提供了防控(Security Collaborator)的功能模块,提供了 UTM 网关连接到监控中心的接口,以便实现全网内通过监控中心的管理员统一管理和配置多台 UTM 网关的安全模块,及时查看和监控每一台 UTM 网关的安全状态和运行信息。防控模块开启之后,会连接到远端的监控中心,监控中心通过内置在 UTM 网关中的探针获取系统状态信息、流量信息和版本信息,用于进行整体的设备状态显示。同时以分组的方式管理设备,以组为单位进行远程统一配置、升级等操作,并可以将管理的 UTM 网关按照一定的策略进行组织,实现设备之间的作战,让威胁在源头得到控制。

9.13.2　配置说明

防控模块的界面如图 9-47 所示,在单击“设置”按钮之后,会进入防控的配置页面。如图 9-48 所示,在基本设置界面中,需要用户提供所要连接的监控中心的 IP 地址,在模块开启模式下,会自动连接到远端的监控中心,将当前 UTM 的每个安全应用和服务模块的运行状态及其他安全信息传输到监控中心,以便集中显示,同时接收远端监控中心发回的配置指令,进行相应的配置。



图 9-47　防控模块界面

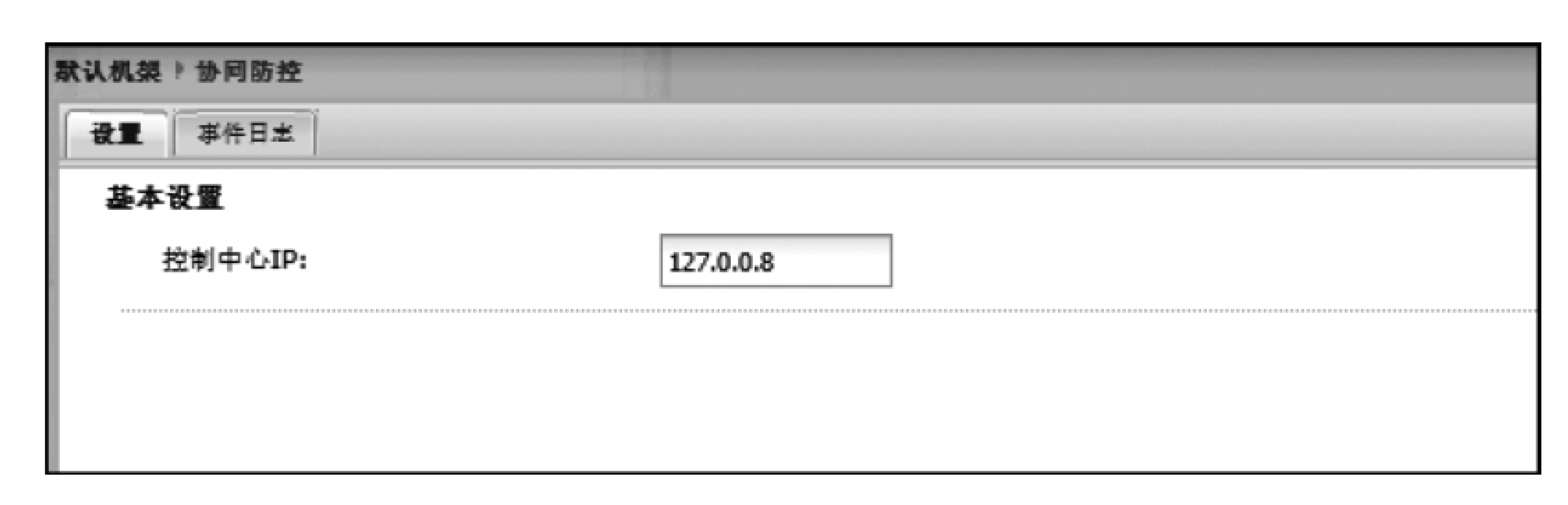


图 9-48　防控基本设置界面

9.14 报　　告

9.14.1　功能介绍

Untangle UTM 可以为用户提供丰富的数据报告。Untangle UTM 提供报告的方式有三种:在线报告、邮件通知和 CSV 文件。在线报告比较详细,用户可以通过 Web 浏览相关数据,并可以打开 CSV 文件详细查看;通过邮件发送摘要报告,其主要是关于网络流量的信息。

9.14.2 配置说明

如果需要在线查看报告，首先需要打开“启用外部报告查看”，操作如下，在主界面右边框中单击“配置”按钮，而后单击“管理”项进入管理配置页面，在“外部管理”区域勾选“启用外部报告查看”，具体可参看 2.4.3 节。

在主界面上单击“报告”功能模块栏上的“设置”按钮，进入配置界面，如图 9-49 所示。



图 9-49 查看报告

在“状态”选项卡下单击“查看报告”按钮，即可打开如图 9-50 所示的“报告”显示页面。该页面默认显示的是系统数据摘要，用户可以在左边框内选择相应的功能模块来查看对应的数据报告。如果需要查看具体数据，可以单击“关键统计”栏下面的小图标，将会打开相应的 CSV 文件，里面记录着详细数据信息。

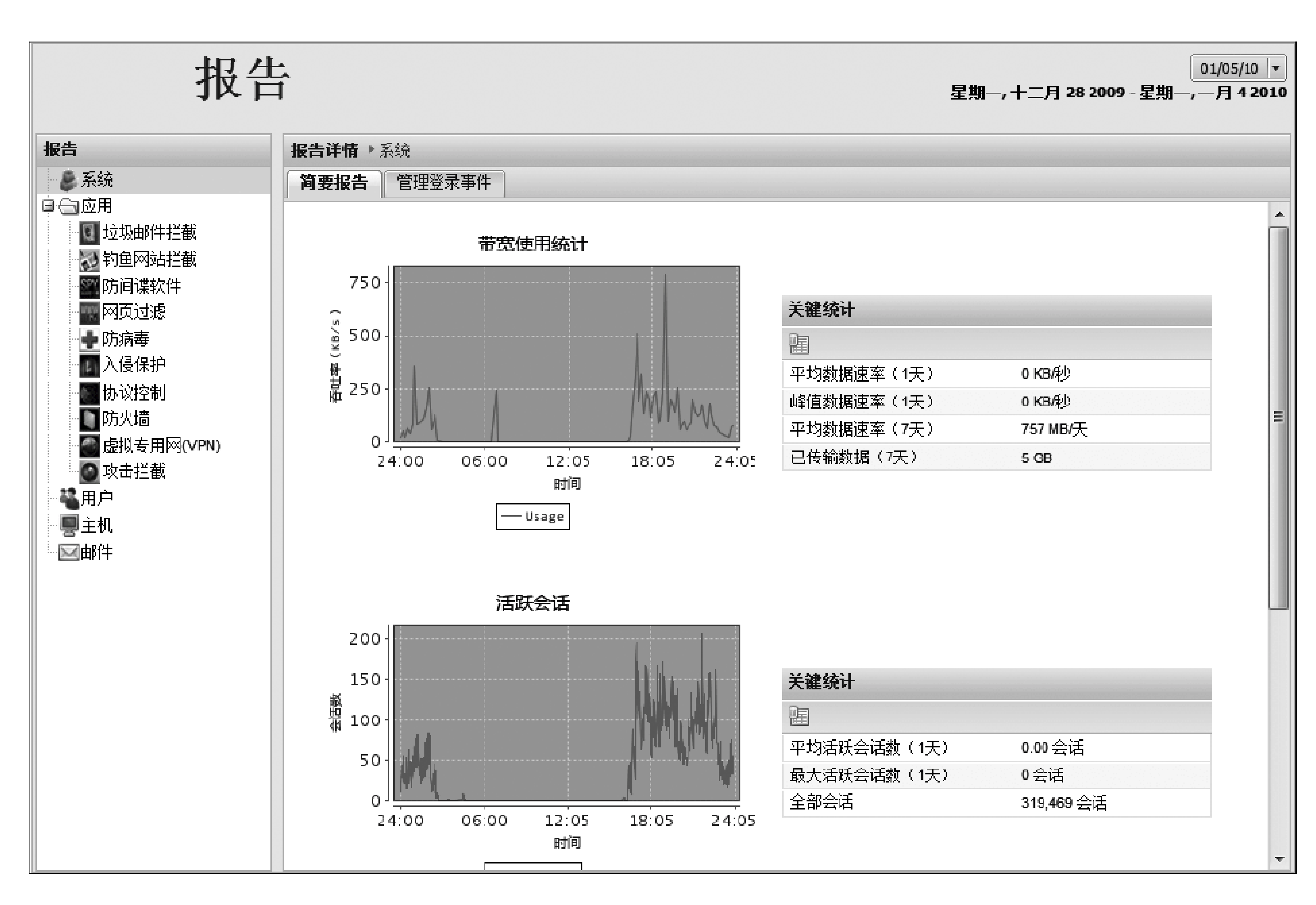


图 9-50 报告主页面

如果希望通过邮件接收报告摘要，可以指定收件人的邮箱地址：单击“生成”选项卡，在“电子邮件”区域下的“收件人”一栏单击“添加”按钮，在弹出框中填写收件人邮箱地址，而后单击“保存”按钮。

系统默认情况下数据保存时限是 7 天，你可以自己设定：在“生成”选项卡的“数据保持”栏中填写希望的时限即可，但数字不能超过 30。

UTM 的报告包含很多 IP 地址，为了增加易读性，可以使用用户名替换内部 IP 地址（受保护的网络中的 IP 地址）。如果使用 DHCP，那就不需要进行这种替换了。请注意，一定不要对外部 IP 地址进行此项操作。要进行替换，操作如下：

单击“报告”下的“名字映射”选项卡，再单击“添加”按钮，在弹出框中填写“名字映射”（内部 IP 地址）和“名称”（用户名）栏，单击“更新”按钮。

注意：查看报告的其他方法：除了通过主界面进入报告页面查看外，还可以通过 URL 直接访问报告页面，方法如下：在浏览器中输入 https://公共地址/reports。公共地址是 UTM 的公共 IP 地址或者主机名。然后输入登录名和密码，就可以查看报告了。

第10章 流量记录

当今互联网上的计算机日益成为网络恶意攻击的目标，随着网络技术的发展，网络破坏技术日益复杂多样，各类组织不得不承担计算机网络攻击带来的大量经济损失及其他损失。

因此，针对计算机网络流量的安全监控的重要性日益突出，急需对网络攻击流量的记录，观察攻击事件“如何”发生，攻击后果“如何”，以应对网络安全事件事前防范、事中查询和事后追踪的需求，提升对网络安全事件的安全防护能力，能够对恶意攻击进行有效遏制。

Untangle UTM 通过流量记录模块制作网络数据镜像，实时记录网络中发生的所有事情，为安全事件发生后的问题分析提供如同黑匣子一样重要的基础数据。通过设置记录的方式，能有效存储 4～7 天的网络流量数据，为安全事件事后分析提供了强大的支持。

此外，可以按照突发安全需求，实时控制流量记录模块聚焦其监管的特殊网域流量，查询相关的突发安全事件网络流量。

Untangle UTM 流量记录模块具有以下特色。

(1) 网络流量的区分归档记录。

(2) 当前流量的即时查询与安全事件报告。

(3) 大容量高速本地记录。

(4) 实时流量查询与显现。

(5) 安全事件回放。

10.1 功能描述

流量记录能够有效记录网包流，感知网络连接，维护一个全局的连接表。为了加快访问速度，该连接表存放在内存中。为了加快查询响应和有效存储大容量网络流数据，流量记录有效利用了内存和硬盘的两级存储空间。同时，流量记录能够有效地监控网络流数据，以网络连接的形式组织数据。一个网络连接定义为由 5 元组刻画的 TCP 或 UDP 的一组网包（源 IP 地址，目的 IP 地址，源端口，目的端口，4 层协议号）。

流量记录可采用对连接传输的大小进行阻断或不阻断的方法，来防止过量消耗存储空间和网络流的保真度，以提高记录流的时效性和全面性。对于非 TCP 和 UDP，可以设置源端口和目的端口无效，其连接标识符设置为 0。这里连接是双向的，即前向和反向的视为两个单向连接。对于每个连接相关的状态变量包括字节数、连接发送的网包数、连接最后一个网包的时戳，以及一些标志位。连接表中的一些标志位用来取消一个连接的阻断，或标识该流已被控制台订阅(Subscribe)。

图 10-1 显示了 Untangle UTM 的流量记录模块面板。

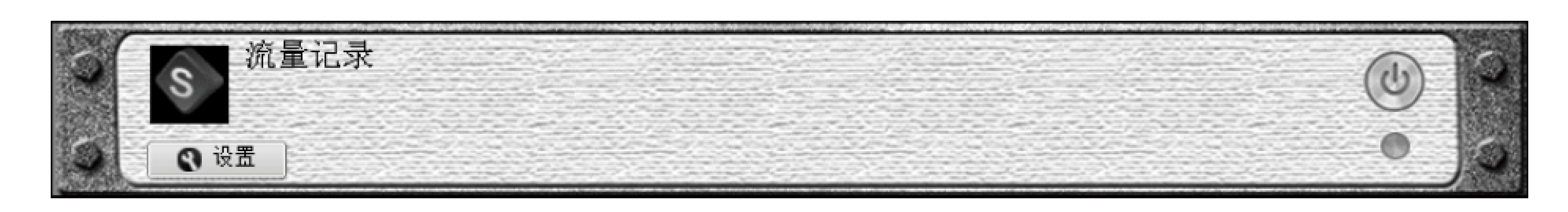


图 10-1　流量记录模块面板

10.2 配置设置

10.2.1 记录网络流量的类型

针对所有网络流量，可设置网络数据记录模式为无截断模式和截断模式(固定记录网络流的指定大小)。

10.2.2 特殊网络流量

(1) HTTP Web：网页浏览。
(2) POP3、SMTP、IMAP4 等电子邮件交互协议。
(3) IM(IRC、QQ、MSN)：即时通信消息。
(4) 文件下载(File Transfer Protocol，FTP)。
(5) Peer-to-Peer(P2P)：文件下载分析。
(6) Voice over IP(VoIP)：会话。

10.2.3 指定流量查询

关键词索引有 9 个：ip、srcip、dstip、port、srcport、dstport、connection2，connection3、connection4。这 9 类索引的流量代表基于 TCP/IP 五元组数据记录的由粗到细的网络流量的分类与存储。

如指定查询细粒度的连接 connection4 的流"tcp 166.111.137.20:80 166.111.130.25:88"，这样有利于最终追踪安全事件。

10.2.4 网络流量记录预处理

探针系统前端设有预过滤器(Pre-Filter)，过滤器工作模式有包含式记录和排他性记录两种。包含式记录 (Include)，即专门记录某些特定的流量，而排他式记录(Exclude)，即对某些主机、网络不予记录，进行选择性忽视。

10.2.5 查询流量大小

系统快速查询流量大小(存放在高速缓存)＞1000MB

系统长期档案文件大小(存放在硬盘中)＞900GB

10.2.6 单流记录大小

流的全部内容：设置网络数据记录模式为无阻断模式，是指完整地记录网络流的所有长度。用户也可以根据需要自定义单流记录大小。流量记录的典型系统参数如表 10-1

所示。

表 10-1　流量记录的典型系统参数

系统配置参数	数　值	系统配置参数	数　值
系统内存	＞2GB	FIFO(内存)	＞1024MB
最小存储容量	＞1TB	FIFO(硬盘)	＞900GB
最小硬盘写入速度	＞200Mb/s		

流的前 30KB 内容：设置网络数据记录模式为阻断模式，截断门限为 30KB，是指固定记录网络流的前 30KB 长度流量。

流量记录配置界面如图 10-2 所示。

图 10-2　流量记录配置

说明：探针系统具有阻断模式，阻断的门限决定了网络流量记录的完整性和开销。无阻断模式意味着完整性最高，开销最大；阻断门限越高，完整性越高，开销也越大。

10.3　查询命令

用户通过图形化界面，发布查询命令给流量记录执行查询。流量记录通过执行用户的查询命令，用户可以获取监控的网包内容。查询结果由流量记录生成发送回查询用户，或者以文件形式存放在流量记录硬盘。

查询名称＜query-spec＞由索引名称(大类)加上索引名称的说明部分(具体内容)组成。

索引名称说明内容由＜索引名称具体内容＞指定，比如索引名称为 connection4 的具体内容为 "tcp 166.111.137.20:80 166.111.130.25:88"。

查询标志＜query-flags＞可以用来约束查找。查询标志包括时段 start ＜timestamp＞ end ＜timestamp＞、mem_only 和 subscribe。查询标志可以顺序给出或者组合。

为了查询一段时间间隔中的流量，可由 timestamps(时戳)来指定时间跨度。只有在该时间跨度内网包才被查询返回。通过周期发出更新的时间跨度的间隔可以实时动态获取最新的查询内容。

为了查询响应快，响应内容新，可指定 mem_only，这样查询只查找存储于内存中的索引项(Index Entries)，返回存放在内存环形队列的网包中。mem_only 和 timestamps 时戳可以组合使用，最终结果为两个查询返回的交集。

设置为“订阅”(Subscription)的查询，可以实时将查询内容动态发回给查询端。当前只有索引名称为 connection4 的查询才支持订阅(Subscribe)标识。

查询界面如图 10-3 所示。

图 10-3　流量查询界面

第 11 章　硬 件 定 制

Untangle 总体是基于软件的 UTM 解决方案，通过采用定制化的硬件，可以提升性能，系统的整体可靠性和安全特色更好。

11.1　硬件定制选型

基于 NexCom 工业主板是目前比较适合网络安全的服务器级主板。表 11-1 给出了 NexCom 工业控制主板的一些具体参数。

表 11-1　网络安全平台参数比较

硬件平台	NSEC-G41	NSA-1083	NSA-1088E
CPU	Intel Core 2 Quad / Core(TM) 2 Duo /	Intel Core 2 Duo Processor，Speed up to 2.66GHz	Intel Core 2 Duo
前端总线	800/1066/1333 MHz FSB		800/1066 MHz FSB
芯片组	Intel G41 and ICH7R Chipset	Intel Q965＋ICH8	Intel 945G ＋ ICH7R Chipset
高速缓存	1MB/2MB cache	4MB Cache	1MB/2MB Cache
内存类型	DDR3 1066 with Non-ECC memory，2 DIMM sockets support Max 4GB	2 x DDRⅡ 533/667/800 DIMM，Max 4GB	DDRⅡ 533/667 DIMM Sockets，up to 4GB
网络芯片	Intel 82574L	Intel 82573L PCI-E GbE,; PCI-32 GbE,	Intel 82571EB
网络类型	Max 8 ports	8 ports copper	4 Copper ＋ 4 SFP
硬件 Bypass	4 pairs	3 Pairs Dual Latch	2 pairs

为了支持特定的硬件(如网卡)，需要在 Untangle 系统中重新编译安装相应的硬件驱动程序。详细流程可参考本书下篇内容。

11.2　业务安全功能

11.2.1　旁路直通功能

在硬件主板支持的情况下，可以实现基于硬件的旁路功能。在具有硬件旁路的功能支持下，默认情况下，Untangle UTM 会在意外断电情况下保持网络连接的旁路是直通的，因此不会影响正常的网络使用。

为了达到控制效果，Untangle UTM 主要以桥接方式部署在网络出口，由此可能会带来系统的潜在单点故障风险。为了保证高可用性，解决方案就是使用硬件主板网口的旁路功

能。当故障发生时，Untangle UTM 自动接通网络原有链路，保证网络正常工作。

基于 NexCom 工业主板安装的 Untangle UTM 设备，在软件配置下，实现硬件旁路功能。

以基于 NSEC-G41 工控主板的 Untangle UTM 为例，实现了断电旁路直通和用户程序控制开关旁路直通功能。有 8 个千兆接口，支持四路旁路直通，每对网口组成一组。比如 Eth0 和 Eth1 组成一组，组号为 0；Eth2 和 Eth3 组成一组，组号为 1，Eth4 和 Eth5 组成一组，组号为 2，Eth6 和 Eth7 组成一组，组号为 3。当遇到紧急情况时，需要开启硬件旁路功能，只需长按 Bypass 按钮（见图 11-1）大于 2s，当听到轻微的跳闸声后，继电器进行了切换，系统转变为旁路模式，同时，旁路指示灯将亮起以显示四路旁路状态。解决问题后，可再次长按 Bypass 按钮（大于 2s），Untangle UTM 将恢复正常工作。



图 11-1　一键 Bypass 按钮示意图

同时，Untangle UTM 还提供了软件旁路功能，即使在安全组件功能受攻击或软件意外失效的情况下，网络使用依然不受安全组件失效的影响，提供设备的整体可用性。

Untangle UTM 提供的软件旁路直通的独有特色功能，配置界面如图 11-2 所示。通过控制界面的软件旁路直通功能，能够有效防止因软件安全组件故障造成的网络问题，保证网络正常工作。

图 11-2　一键 Bypass 按钮配置界面

11.2.2　硬件看门狗

以 Untangle UTM 为例，为了能够提供更高的可靠性，除了旁路直通功能外，还具有看门狗功能（Watchdog），看门狗功能初始设计的基本用途是控制系统重启，可以在硬件中设定，也可以通过监护程序控制，监护程序执行的是“喂狗”动作，如在一个时间周期内，不断给定时器刷新时间，看门狗就不发出重启指令；如果系统超时，如系统宕机时，看门狗不能正常工作，则发出系统重启指令。

11.2.3　软件看门狗

以 Untangle UTM 为例，为了进一步提供安全功能的可靠性，除了旁路直通功能和硬件看门狗功能，还实现了软件看门狗功能。软件看门狗初始设计的基本用途是监控各安全功能，如反病毒、IPS、垃圾邮件过滤等安全功能。一旦出现异常，如系统资源消耗过多、安全模块停止等，可以根据策略，将该安全功能重新启动，从而避免了因安全功能模块失效而

导致网络业务中断的问题，保障了网络业务的高可用性。软件看门狗配置界面如图11-3所示。



图11-3　看门狗按钮示意图

第 12 章 操 作 实 例

12.1 故障排除与应急处理

12.1.1 网络中出现大量垃圾邮件

1. 检查垃圾邮件拦截日志

单击“垃圾邮件拦截”→设置→事件日志命令，出现如图 12-1 所示垃圾邮件事件日志。

默认机架 ▸ 垃圾邮件拦截　　出现大量高垃圾评分邮件

电子邮件　事件日志　防垃圾陷阱事件日志

时间戳	动作	客户端	主题	收件人	发件人	垃圾评分	服务器
2009-12-29 5:13:25 pm	通过邮件	192.16...	Pecker enlargement free tri...	chy@...	oceansid...	0.1	203.8...
2009-12-29 5:13:25 pm	标记	192.16...	[SPAM] Facebook account ...	hnvs...	update+j...	10.8	203.8...
2009-12-29 5:13:25 pm	标记	192.16...	[SPAM] Facebook Update T...	jezha...	update+ft...	8	203.8...
2009-12-29 5:13:24 pm	标记	192.16...	[SPAM] Lilly giving Viagra a...	chy@...	allelejo@y...	8.3	203.8...
2009-12-29 5:13:24 pm	通过邮件	192.16...	Free Sample our penis enla...	dy@e...	morley3K...	0.8	203.8...
2009-12-29 5:13:23 pm	通过邮件	192.16...	[SPAM] =?koi8-r?B?QSBia...	yunn...	spaceshi...	2.9	203.8...
2009-12-29 5:13:23 pm	标记	192.16...	[SPAM] Facebook Update T...	sunr...	update+c...	9.7	203.8...
2009-12-29 5:13:23 pm	标记	192.16...	[SPAM] Tired of paying an a...	sunr...	marierich...	10.5	203.8...
2009-12-29 5:13:22 pm	标记	192.16...	[SPAM] Visitor zhejiang's p...	zhejia...	ifahasu70...	5.7	203.8...
2009-12-29 5:13:22 pm	标记	192.16...	[SPAM] new login system	sichu...	update+x...	8	203.8...
2009-12-29 5:13:22 pm	通过邮件	192.16...	[SPAM] Watch dealer online	jilin@...	quality.lon...	3.5	203.8...
2009-12-29 5:13:21 pm	标记	192.16...	[SPAM] Visitor sichuan's pe...	sichu...	uusuycou...	5.7	203.8...
2009-12-29 5:13:21 pm	通过邮件	192.16...	Gain a massive 3 inches wi...	dy@e...	bestirring...	0.1	203.8...
2009-12-29 5:13:21 pm	标记	192.16...	[SPAM] Facebook account ...	sichu...	update+o...	11.5	203.8...

所有事件　刷新　自动刷新　页 8　页共 17 页

移除　取消　保存

图 12-1　垃圾邮件事件日志

如从日志中发现大量邮件垃圾评分较高，被判定为网络内出现大量垃圾邮件。

2. 配置 SMTP 扫描(扫描内网发往外网的邮件)

单击“垃圾邮件拦截”→设置→电子邮件命令，出现如图 12-2 所示邮件拦截设置。

其中扫描强度越高(即阈值较低)，则越多的邮件将被判定为垃圾邮件。如阈值为 3.0，垃圾评分超过 3.0 的邮件被判定为垃圾邮件。相反，阈值越高，则相对较少的邮件被标记。

另外，大量垃圾邮件引起邮件扫描引擎负载迅速升高。为了保持整体性能，可以调节 CPU 负载限制和并发扫描引擎数。从而使得本模块不至于占用过高系统负载，影响其他安全应用的正常运转。一般推荐 CPU 负载限制为 10%～15%，并发扫描数限制为 15%～30%。用户根据具体情况可适当调节此参数。

12.1.2 网络拥塞应急处理

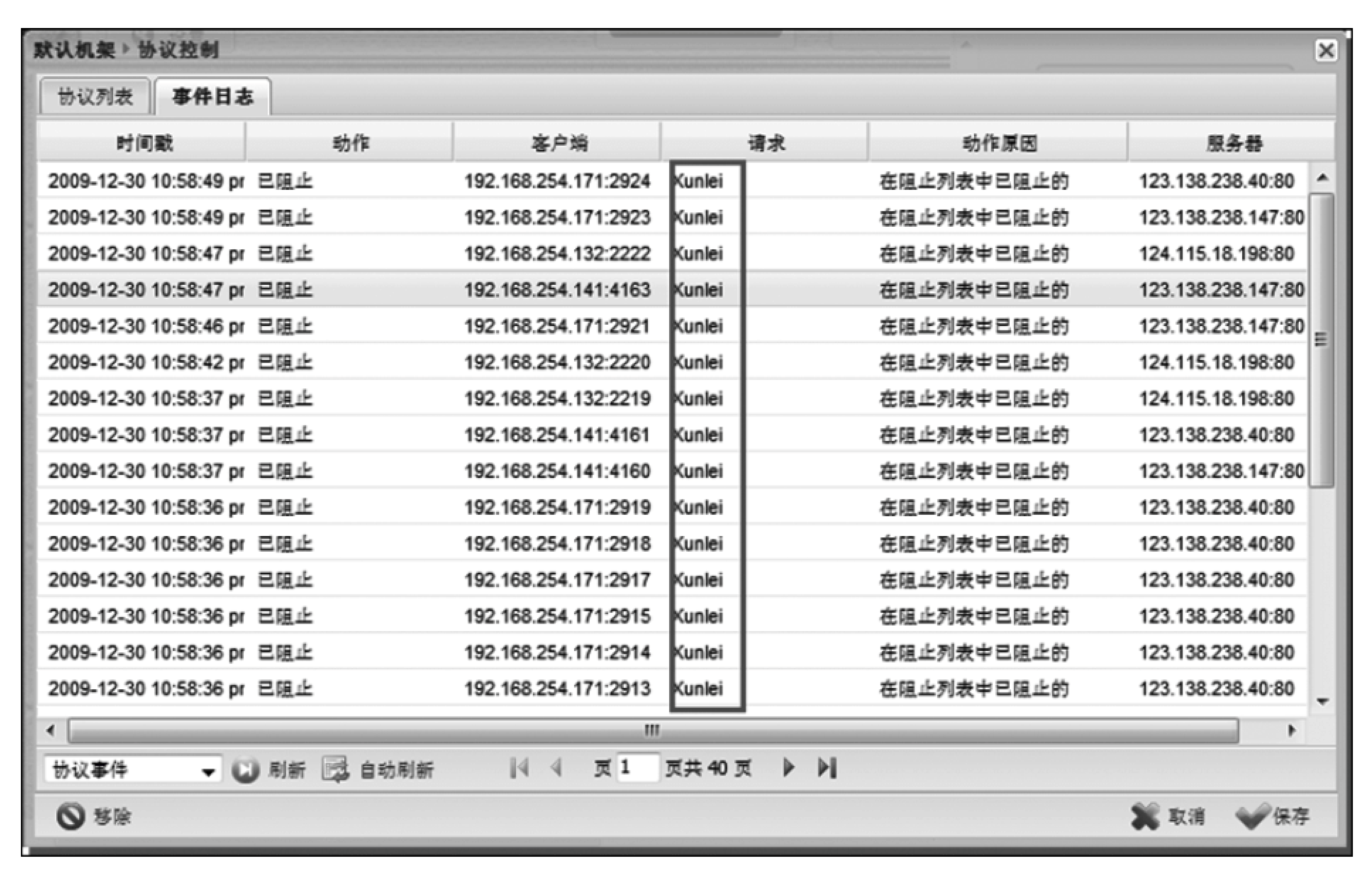
网络中出现的大量 P2P、BT 流量占用有限带宽，影响关键业务使用。



图 12-2　邮件拦截设置

1. 协议控制事件日志

单击"协议控制"→"设置"→"事件日志"命令，出现如图 12-3 所示协议控制事件日志。



时间戳	动作	客户端	请求	动作原因	服务器
2009-12-30 10:58:49 pr	已阻止	192.168.254.171:2924	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:49 pr	已阻止	192.168.254.171:2923	Xunlei	在阻止列表中已阻止的	123.138.238.147:80
2009-12-30 10:58:47 pr	已阻止	192.168.254.132:2222	Xunlei	在阻止列表中已阻止的	124.115.18.198:80
2009-12-30 10:58:47 pr	已阻止	192.168.254.141:4163	Xunlei	在阻止列表中已阻止的	123.138.238.147:80
2009-12-30 10:58:46 pr	已阻止	192.168.254.171:2921	Xunlei	在阻止列表中已阻止的	123.138.238.147:80
2009-12-30 10:58:42 pr	已阻止	192.168.254.132:2220	Xunlei	在阻止列表中已阻止的	124.115.18.198:80
2009-12-30 10:58:37 pr	已阻止	192.168.254.132:2219	Xunlei	在阻止列表中已阻止的	124.115.18.198:80
2009-12-30 10:58:37 pr	已阻止	192.168.254.141:4161	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:37 pr	已阻止	192.168.254.141:4160	Xunlei	在阻止列表中已阻止的	123.138.238.147:80
2009-12-30 10:58:36 pr	已阻止	192.168.254.171:2919	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:36 pr	已阻止	192.168.254.171:2918	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:36 pr	已阻止	192.168.254.171:2917	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:36 pr	已阻止	192.168.254.171:2915	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:36 pr	已阻止	192.168.254.171:2914	Xunlei	在阻止列表中已阻止的	123.138.238.40:80
2009-12-30 10:58:36 pr	已阻止	192.168.254.171:2913	Xunlei	在阻止列表中已阻止的	123.138.238.40:80

图 12-3　协议控制事件日志

这时会发现系统中存在"迅雷"等大量 P2P 应用，它们严重挤占系统带宽。

2. 阻断迅雷协议

单击"协议控制"→"设置"→"协议列表"命令，出现如图 12-4 所示界面。

通过翻页找到迅雷协议（其中分类是对协议自身做粗粒度的聚合，如将迅雷分类为 P2P 应用），勾选其阻止和日志两项，从而阻断网络中的迅雷协议的网流。

用户根据自身需求可以自行添加或者修改协议，如图 12-5 所示。



图 12-4　查找特定事件

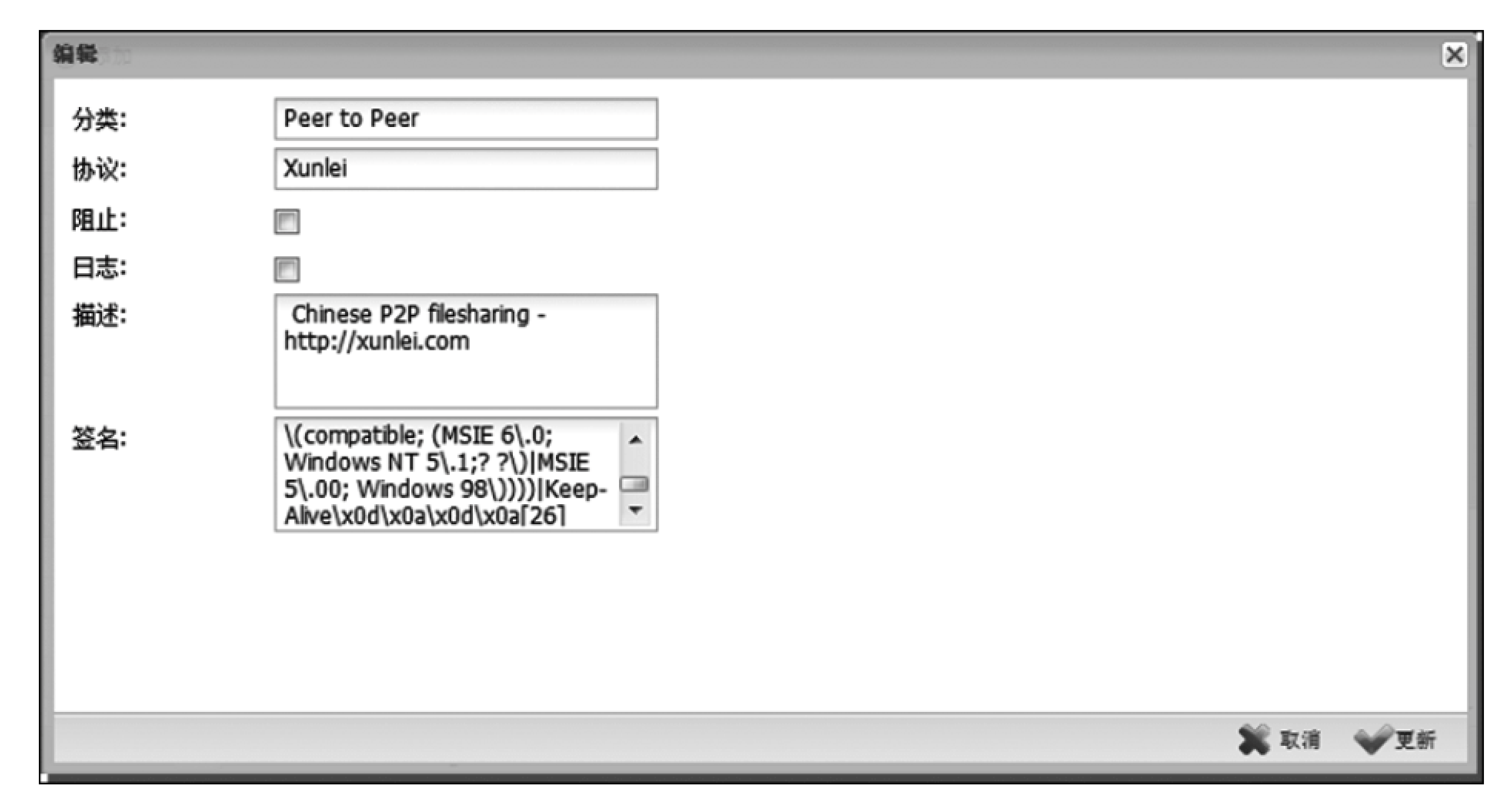


图 12-5　自定义协议

如果用户要阻断或者放行某种协议与此类似。

12.1.3　某内网 IP 地址对应正常服务被拦截

对于某内网 IP，如果不希望其流量被意外拦截，可以打开 UTM 的软件旁路功能。假设内网服务所在 IP 地址为 192.168.2.123，单击配置→联网页面右上角的“高级”选项，选择“旁路规则”，如图 12-6 所示。

单击“添加”按钮，增加新的规则，如图 12-7 所示。

在弹出的规则添加页面中，输入规则名字，选中“旁路”复选框，单击“添加”按钮新增条件匹配，选择源地址并输入 192.168.2.123，如图 12-8 所示。单击“更新”按钮，进一步单击“保存”按钮。



图 12-6　旁路规则设置



图 12-7　添加新的规则



图 12-8　编辑规则

12.2 集中监控和管理

12.2.1 集中监控

当前，安全威胁日益复杂多样，攻击手段日益智能化，网络安全设备需要时刻抵御新的安全威胁和攻击，因此对网络安全设备的运营状况进行实时监控有助于快速响应安全事件的爆发，维护网络的正常运营。

此外，实时监控有助于排除如操作员失误、软件错误、环境条件等问题造成的故障。研究表明，尽管网络硬件已经十分可靠，但是计划外故障仍会发生，仅凭可靠的硬件是无法避免这些问题的。即使是因系统维护和安全功能升级而产生的计划内中断，也可能会影响网络业务的运行。因此，采用实时的监测管理系统，是实现可靠、及时的网络安全管理的关键。

Untangle UTM 为用户提供了 UTM 设备安全组件集中式的管理和统一的性能与状态监视，具有以下特点。

(1) UTM 设备各种安全功能(防火墙、防病毒、入侵保护、协议控制、垃圾邮件、广告拦截、网页过滤、防间谍软件等)工作状况监控。

(2) UTM 设备系统资源(网络带宽、处理器负荷、存储器利用率和磁盘利用率等)监控。

(3) 并行服务检查机制。

(4) UTM 设备检测到发生的网络安全事件和问题时，将会发送告警信息给网络管理员(通过 E-mail、短信、用户定义方式)。

(5) UTM 设备日志集中管理与查询功能。

(6) UTM 设备良好的 Web 管理接口。

监控程序采用了服务器——代理的工作模式，监控中心通过部署在每台 UTM 设备上的代理程序与每台 UTM 设备进行安全通信，代理程序中的本地监控插件程序将会根据控制中心的命令，指示 UTM 完成相应的策略管理动作：如返回当前各台 UTM 设备的状态信息和性能数据，使得网络管理员能够掌握当前的网络状况；指示各台 UTM 按照一定条件筛选某一时间段内的安全组件日志信息，并将这些信息汇总到数据中心，使得网络管理员能够针对某种网络安全威胁进行快速响应；实现 UTM 安全组件的程序版本升级和安全策略库更新、策略管理动作的更新。监控中心与 UTM 设备的进行基于 SSL 的安全通信，同时 UTM 设备的代理程序也会验证监控中心的身份，只有来自监控中心的监控请求才会得到 UTM 设备的响应。

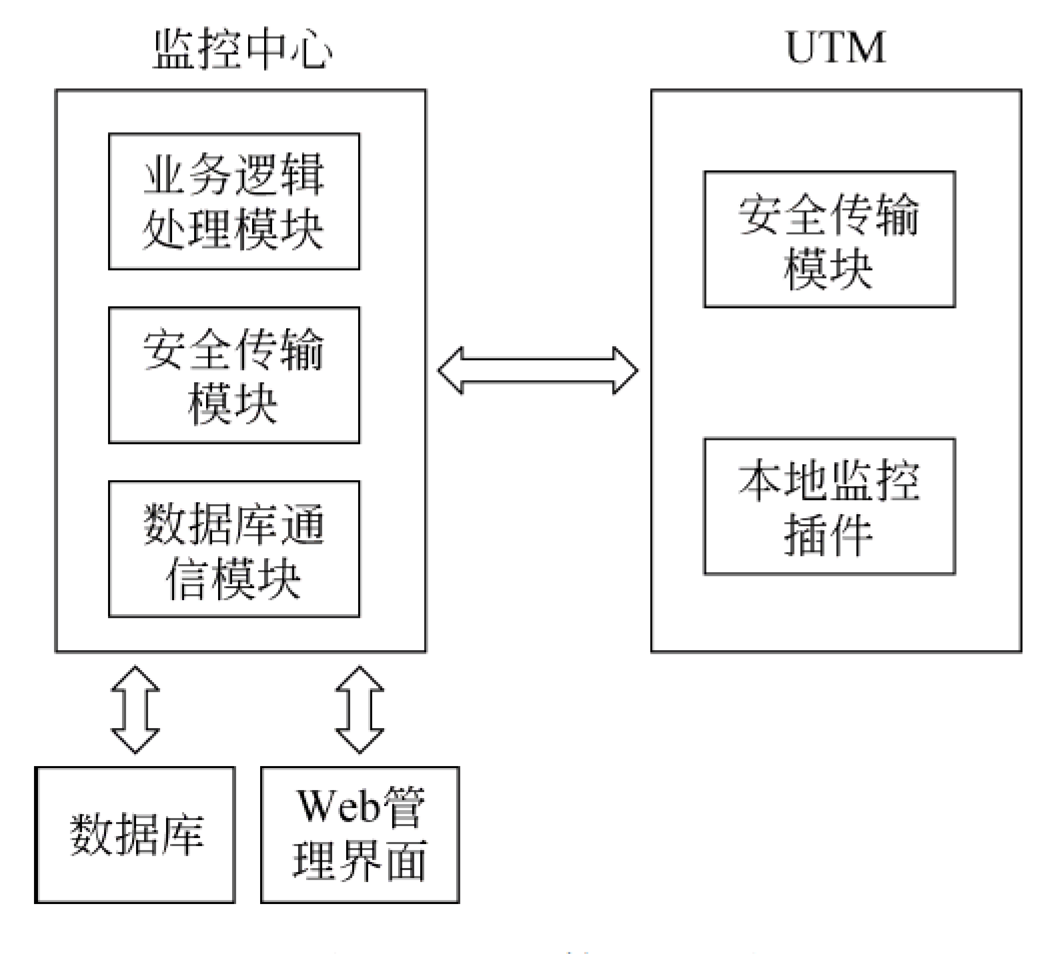


图 12-9 监控原理图

监控原理如图 12-9 所示。

12.2.2 监控 Web 管理界面

当部署了多台 UTM 设备时，可以通过监控中心来对各 UTM 进行集中式的监控和管理，使得网络管理员能够更好地管理网络，应对网络安全威胁。

通过监控中心，能够直截了当地查看各台 UTM 所有安全组件的状态信息和性能数据，使得管理员能够快速和简单地掌握当前的网络环境、发现存在的网络安全威胁。

12.2.3 命令格式与操作

各 UTM 设备通过专用的安全通道进行命令传控通信。控制中心的策略管理动作格式定义如下：

```
define command{
  command_name cmd
  command_line $ USER$ /cmd -H $ UTM_ADDRESS$  -c $ ARG1$  -a $ ARG2$
  }
```

在 UTM 代理程序中所对应的策略管理动作如下：

```
command[cmd]=CMD $ ARG1$ -c $ ARG1$  -a $ ARG2$
```

主要的控制命令按照安全功能、配置、日志、策略、报告类型，共分为 5 个大类，总计 15 项具体功能。

（1）安装与升级功能：Untangle UTM 灵活先进的软件架构使得用户可以对 UTM 设备程序进行部分更新，而不影响正常工作，用户可以根据自身需要安装或者卸载指定的安全组件，实现对于特定用户的定制功能需求，该部分功能包括如下。

① 安装新的安全组件。

② 卸载已安装的安全组件。

③ 对安全组件进行升级。

（2）安全组件操作功能：Untangle UTM 各安全功能组件独立运行和工作，互不影响，因而用户可以对某个安全组件进行独立的操作和控制，使得用户可以根据当前的网络安全状况进行更有针对性的配置和管理，该部分功能包括如下。

① 开启某个安全组件。

② 关闭某个安全组件。

③ 禁用某个安全组件。

（3）管理员系统功能：Untangle UTM 控制中心对网络管理员实行严格的访问登录控制，只有经过身份认证的用户才能够登录，对 UTM 设备进行管理和控制，保证了设备和网络的安全，该部分功能包括如下。

① 获取注册信息。

② 登录监控系统。

③ 登出监控系统。

（4）策略管理功能：Untangle UTM 特有的策略管理功能使得 UTM 设备能够进行工作，共同抵御网络安全威胁。通过加载不同的策略，可以使得 UTM 组发挥不同的安全作

用，因此 Untangle UTM 允许用户添加或者查询当前的策略，该部分功能包括如下。

① 添加新的管理策略。

② 显示当前管理策略。

（5）日志报告系统功能：完备的网络安全日志信息对于网络安全管理至关重要，因此，Untangle UTM 提供了强大的日志记录和管理系统，使得网络管理员能够时刻掌握当前网络的最新状况和突发的网络安全事件，该部分功能包括如下。

① 打开日志功能。

② 启用日志功能。

③ 准备日志报告。

④ 生成日志报告。

表 12-1 给出了具体控制命令类型与具体参数。

表 12-1　控制命令类型与具体参数

名　　称	参　　数
功能命令	CLI active function-name CLI deactive function-name CLI update CLI upgrade
配置命令	CLI active CLI deactive CLI upgradable CLI uptodate
功能控制命令	CLI list function-name [参数] CLI start function-name CLI stop function-name CLI kill function-name
管理命令	CLI who CLI getAdmin CLI passwd [-add | -del] login [password]
CLI 命令	CLI shutdown CLI serverStats
策略命令	CLI addPolicy CLI delPolicy CLI listPolicy
报表命令	CLI enableReporting CLI prepareReports [args] CLI startReports CLI stopReports
日志命令	CLI startLog CLI stopLog CLI restartLog CLI clearLog

第13章　部署使用

Untangle UTM 可以参照使用场景来定制鲜明特色的功能，制定适合行业使用的相关的解决方案。

13.1　金融证券银行业

这些行业对数据访问和业务的安全性要求很高，Untangle UTM 具有的 VPN 功能和高性能网络访问控制（NAC），为这些行业业务的安全提供了有力的保障；另外，在金融监管方面，Untangle UTM 独特的会话代理功能和数据记录功能，能对金融交易的数据进行有效的审计和追踪能力，增强了监管能力。

13.2　教　育　业

中小学环境中用户主要以学生为主，互联网的接入为教学提供了丰富的资源，同时为了保护未成年人，Untangle UTM 具有 Web 过滤，IM 和 P2P 及网络游戏协议控制功能，对防止学生浏览、接触不良信息等，起到积极作用。

13.3　政府办公环境

目前电子政务的普及，对政府办公网络的可信性提出了更高的要求。Untangle UTM 提供了防火墙功能（NG）、入侵保护功能（IPS）、抗 DDoS 功能和反垃圾邮件功能等，对防御和阻止恶意流量攻击，建立可信的网络环境提供了有力保障。

13.4　电信运营环境

当电信运营商接入网络环境中，因为承载的用户的网络行为复杂，网络流量和协议应用丰富，Untangle UTM 具有带宽管理、P2P 协议识别和控制、流量优先级管理，能够有效地保障付费用户的网络用户体验，限制 P2P 协议的流量，对维护良好的电信运营环境提供了有力保障。

13.5　中小企业环境

Untangle UTM 提供了防火墙功能（NG）、入侵保护功能（IPS）、抗 DDoS 功能、反垃圾邮件功能、防病毒功能等，为全面防御和阻止恶意流量攻击，建立可信的网络环境提供了有力保障。

第 14 章　问题解答

1. 防间谍软件

问：用户抱怨无法连接 xxx.com 站点，并且在时间日志中始终显示。我该如何解除对 xxx.com 站点的阻止？

答：你可以在“通过列表”中为站点 xxx.com 增加一条规则。

问：我希望在我的网络中阻止某个 ActiveX，但我的一个商业伙伴站点需要该 ActiveX 时，我该如何配置？

答：首先，禁用所有 ActiveX 控件。然后，通过将商业伙伴的域名添加到“通过列表”中来解除对你的商业伙伴的这个限制。

2. 钓鱼网站拦截

问：在对接收自 IMAP 的钓鱼邮件进行标记配置时，邮件主题只在我单击邮件时改成了“钓鱼”字样，为什么？

答：大多数 IMAP 客户端首先获取邮件摘要信息(如主题、发送者)，因此用户能够看到一些关于邮件概要信息。只有当用户选择(单击)该邮件后，才会真正从服务器上接收下来。这时 Untangle UTM 才能够进行扫描，有些邮件客户端并不能检测到邮件主题的改变，从而也就没法更新其概要信息。

3. 广告拦截

问：广告拦截规则如何更新？

答：广告拦截的规则列表包含了链接至广告的站点，以及非广告站点中的广告链接。随着一些厂家发布新的广告，该列表也会被更新。Untangle UTM 会收集这些更新并将其加入到更新列表中。

4. 防火墙

问：为什么 Untangle UTM 防火墙组件默认规则是全部通过？

答：当 Untangle UTM 为路由模式时，内外网是 NAT 架构。NAT 可以阻断大部分威胁；当 Untangle UTM 为桥接模式时，它通常已经在一个防火墙之后，该防火墙也能阻断大部分威胁。

问：Untangle UTM 支持端口映射吗？

答：端口映射(端口转发)的配置在配置→网络→端口转发中。

问：通过地址转换后，在防火墙规则中应该使用转换前地址还是转换后地址？

答：Untangle UTM 防火墙的地址匹配总是匹配含更多信息量的地址。如果某网络地址从 192.168.*.* 转换为 1.2.3.4，对于进出该网络的流量，Untangle UTM 防火墙会匹配 192.168.*.*。从会话层面来说，即为源地址是匹配转换前地址，目标地址是匹配转换后地址。

5. 入侵保护

问：为什么不对大部分的规则都实施默认阻挡？

答：因为除了恶意行为以外，这些规则同样能够阻挡非恶意的行为。因此，为了方便用户，Untangle UTM 对每条规则和网络状态进行评估，来确定合适的默认规则。

6. 防病毒

问：如果我使用了 Untangle UTM，是否还需要在网络上的每台单机上安装防病毒软件？

答：如果你已经启用了 Untangle UTM 上的防病毒模块，那么 Untangle UTM 将会扫描所有的通过 Untangle UTM 的 E-mail 流量，实现对于用户的第一层保护。但是并不能保证所有的流量都通过 Untangle UTM，所以 Untangle UTM 建议用户增加另一层保护，为所有桌面电脑和笔记本电脑安装防病毒软件。

7. 网络不稳定

问：使用 Untangle UTM 过程中，为什么会出现网络不稳定现象？

答：出现网络不稳定的情况，可打开硬件 Bypass 功能，详见 11.2.1 节。如果现象依旧，说明是网络自身的问题，很可能是 ISP 或者教育网络接入存在路由器热备份、宕机，或者病毒爆发或 DDoS 等安全攻击现象。请联系 ISP 或者教育网管理员。

如果现象依然存在，可打开软件 Bypass 后，详见 8.3.2 节中的流量旁路配置。如果网络依然不稳定，很可能是 Untangle UTM 受攻击或者网络内部有病毒爆发。如果现象消失，可检查日志和异常报告，很可能是网络内的病毒和垃圾邮件比较多的缘故，UTM 经常性检查出恶意威胁和攻击异常，导致正常用户上网比较慢，这时需要对内网的机器严格盘查。

展　　望

互联网安全平台的配置和操作是一件非常耗时耗力的事情，但是从中可以学习到实际工程项目的经验，实际接触现实网络中各种各样的问题，提高分析与解决问题的能力与素养。

下　　篇

网络安全平台开发

第 15 章　开源 UTM 系统介绍

开源 UTM 系统主要有 pfsense、Endian Firewall 和 Untangle 系统，它们分别基于 FreeBSD、Fedora Linux 和 Debian Linux 操作系统开发，其中功能又以 Untangle 系统最为丰富。目前 Untangle 版本为 10.0，且还在不断开发中。Untangle 10.0 增加了 httpsInspector 组件，能够对 Https 协议分析。

Untangle UTM 网关系统以其功能丰富和配置简单友好，在开源网关系统中占有率很高，在中小企业中有广阔的应用前景。同时 Untangle 系统是在 Linux 基础上，是集成了开源安全功能模块的软件系统，包括网络处理、虚拟安全功能平台，Web 管理配置界面，系统结构比较复杂，因此需要对其做剖析，以利于深入掌握和进行二次开发。

Untangle 创始人兼 CEO 是 Dirk Morris，毕业于美国卡耐基梅隆大学计算机系。另一位创始人兼 Chief Architect 是 John Irwin。Untangle 系统经过多年的发展，功能日益丰富，其安全组件生命周期管理非常灵活，能够按需下载、装载、运行，关闭和卸载。新的安全组建开发简洁，高效，如新增 facebook 等 SNS 网络的安全插件(Saveface)，这都依赖于其系统架构的优良设计。其专利技术有 Virtual Pipeline Technology(虚拟流水线技术)，能够有效降低时延。

第 16 章 Untangle 系统基本原理

16.1 防火墙技术分类

防火墙技术主要分为 4 类。

(1) 网包过滤(Packet Filter)。

(2) 应用级网关(Application Gateway)。

(3) 电路级网关(Circuit-level Gateway)。

(4) 网包状态检查(Stateful Packet Inspection,SPI)。

16.1.1 网包过滤

网包过滤主要由一个入端口、一个出端口和一组规则组成。规则库决定是否允许网包进入信任网络。一般是基于源地址和目的地址、应用、协议以及每个 IP 包的端口来做出通过与否的判断。防火墙检查每一条规则直至发现包中的信息与某规则相符。如果没有一条规则能符合,防火墙就会使用默认规则,一般情况下,默认规则就是要求防火墙丢弃该包。其次,通过定义基于 TCP 或 UDP 网包的端口号,防火墙能够判断是否允许建立特定的连接,如 Telnet、FTP 连接。

16.1.2 电路级网关

电路级网关是在 OSI 模型的会话层(Session)过滤网包,这样比包过滤防火墙要高二层。电路级网关允许用户通过身份验证后穿过网关访问服务。电路级网关的工作方式是对信任网络发起的请求进行中转。中转过程中源 IP 地址转换为电路级网关的地址。电路级网关控制受信任的网络主机与不受信任的主机间的 TCP 握手信息,这样来决定该会话是否合法。

电路级网关还提供一个重要的安全功能——代理服务器(Proxy Server)。代理服务器是设置在网关的专用应用级代码。这种代理服务准许管理员允许或拒绝特定的应用程序或一个应用的特定功能。包过滤技术和应用级网关是通过特定的逻辑判断来决定是否允许特定的网包通过,一旦规则匹配,防火墙内部网络和外部网络之间就直接连通。这就引入了代理服务的概念,即防火墙内外网络主机应用层的连接(Connection)由两个终止于代理服务的连接来实现,这样实现了防火墙内外网络的隔离。同时,代理服务还可引入过滤、Web 过滤、协议控制和报告等新的安全功能。

16.1.3 应用级网关

应用级网关对信任网络用户来说是一个服务器,对目标服务器来说是一个客户端,它提供了比网包过滤机制更高的安全性,可以对协议进行过滤,对内部的网络服务进行保护。应

用级网关能够检查进出的网包，通过网关复制传递数据，防止在受信任服务器和客户机与不受信任的主机间直接建立联系。

应用级网关能够解析应用层的协议，能够做协议控制，并做审计和记录，针对特别的网络应用服务协议，能够对其分析并形成相关的报告。在实际工作中，应用网关一般对每一种协议需要相应的代理软件，使用时工作量大，效率不如网络级防火墙。应用级网关有较好的访问控制，是目前最安全的防火墙技术，但实现困难。

16.1.4 网包状态检查

该技术结合了包过滤防火墙、电路级网关和应用级网关的特点。它同网包过滤防火墙一样，按照规则检查防火墙能够通过的 IP 地址和端口号，过滤进出的网包。它同电路级网关一样，能够检查 SYN 和 ACK 标记和序列号是否正确。也像应用级网关一样，可以在应用层上检查网包的内容，查看这些内容是否能符合企业网络的安全规则。

规则检查防火墙虽然集成前三者的特点，但是不同于一个应用级网关的是，它并不打破客户机/服务器模式来分析应用层的数据，它允许受信任的客户机和不受信任的主机建立直接连接。规则检查防火墙不依靠与应用层有关的代理，而是依靠算法来识别进出的应用层数据，这些算法通过已知合法网包的模式来比较进出网包，这样比应用级网关代理在过滤网包上更有效。

16.2 Untangle 系统

Untangle 系统是一个具有以上 4 种防火墙技术的 UTM 设备，以电路级网关为主，并兼有应用级网关特点的 UTM 设备，同时具有网包过滤和网包状态检查功能，实现内外网络的隔离。

在会话层，Untangle 系统是一个电路级网关。Untangle 系统是基于流处理的代理模式。以代理的模式将一个会话拆分为两段，Untangle 本身作为一个中间代理，与连接的双方分别建立连接。代理模式的特点如下。

(1) 通过 UTM 的会话被拆分为源端(src)到 UTM 和 UTM 到目的端(sink)两个会话，拆分过程对于原会话双方是透明的。

(2) 在 UTM 中，根据会话的各种基本属性，以及应用层协议内容对会话做出相应的策略判断：通过或是阻断。

(3) 安全应用模块以串联的方式接入 UTM 的处理流程中，可以动态启用或停止，加入新模块或删除旧模块。

在高级安全应用处理层面，Untangle 系统是一个应用级网关。同时 Untangle 系统也采用 Linux 系统的 IPtables 功能，而 IPtables 模块具有网包过滤防火墙功能。此外，Untangle 系统也是一个网包状态检查防火墙，具有自身攻击防御和 DDoS 防御功能。

第 17 章　Untangle 系统功能

17.1　Untangle 系统架构

Untangle 系统是一种分层的架构，分为应用层、应用框架层、系统运行库层和 Debian Linux 层 4 个层次，从下往上依次为 Debian Linux 系统，支持系统和运行时，Untangle 应用框架和 Untangle 安全应用。架构图如图 17-1 所示。

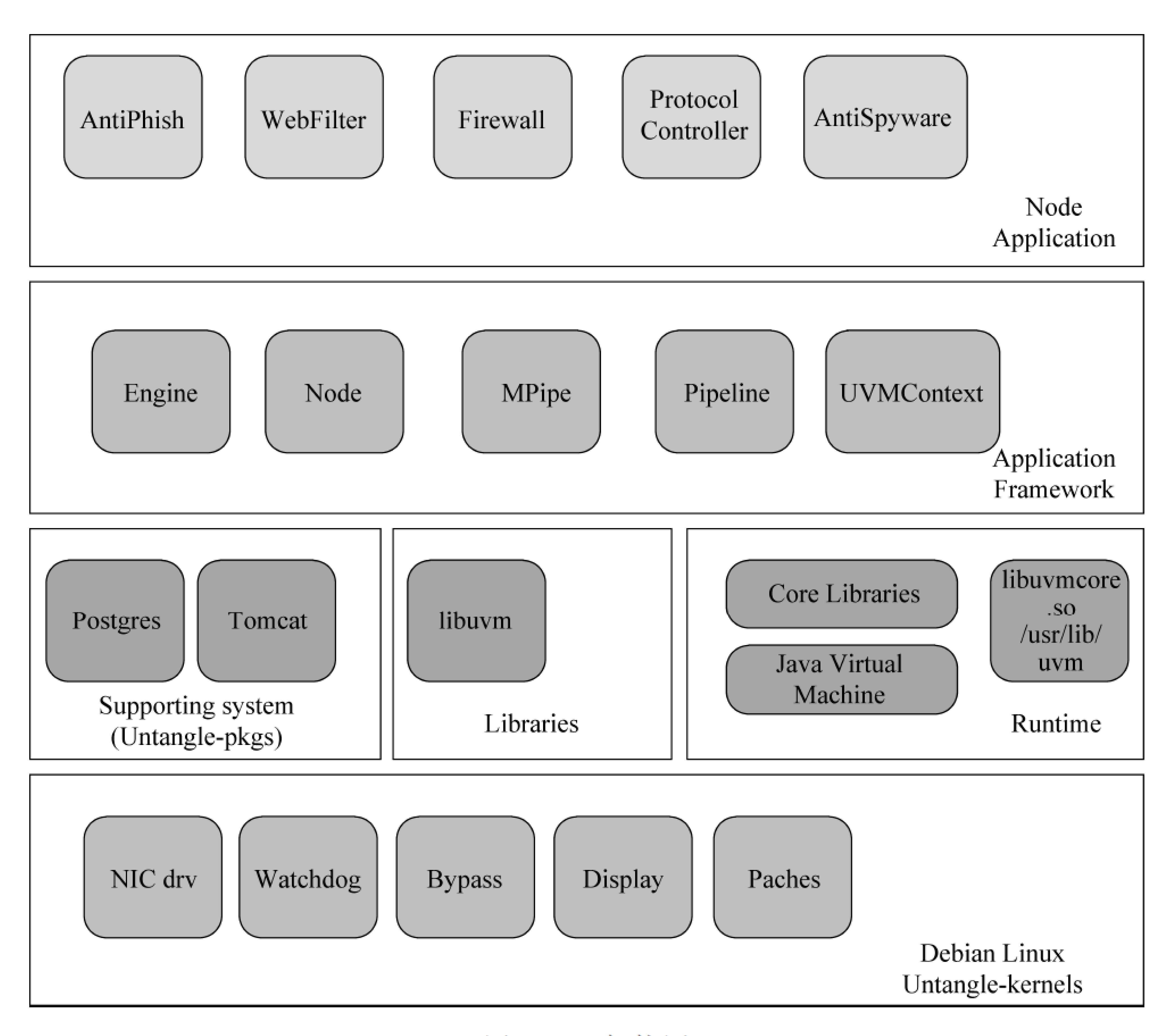


图 17-1　架构图

从功能上看，Untangle 系统主要分为以下几大部分。

（1）Debian Linux 基本系统。

（2）Untangle 系统文件。

（3）Untangle 虚拟机。

（4）Linux-UVM 结合部。

（5）安全主功能系统。

（6）Web 界面系统。

(7) 数据库系统。
(8) 高可用性。
(9) 自身安全。

17.2 Debian Linux 基本系统

Untangle 系统基于 Debian Linux 版本,目前是 Debian 5。Untangle 作为网关系统,自身的稳定性和安全性至关重要,而 Debian Linux 系统以其稳定性和安全性著称,是当前网关服务器的首选。因此基于 Debian Linux 的架构是 Untangle 系统的基本设计选项。但是也需要注意,由于 Debian Linux 系统的稳定性要求,导致测试时间比较长,版本更新周期长。为了克服以上缺点,Debian Linux 的衍生版本 Ubuntu Linux 的目标就是版本更新速度更快,功能更丰富。

Debian Linux 系统采用 deb 包管理系统,通过 APT 包管理工具管理,因此 Linux 内核、内核驱动、应用均以 deb 安装包的形式存在(扩展名为 deb)。

Untangle 系统同时集成 Linux 系统安全功能模块,如防火墙模块 NetFilter/IPtables、反病毒模块 ClamAV、流量控制管理模块 TC、反垃圾邮件模块 SpamAssassin、反广告件模块 Adblock、安全通道 Stunnel、入侵检测 Snort 系统规则集、应用层协议识别 L7-filter。另外也集成了 Apache Web 系统、PostgreSQL 数据库、Monit 自身监管系统等。

查看当前安装的 deb 包的命令有 aptitude 命令和 apt 命令,所有 deb 包均存放在以下目录中:

```
/var/cache/apt/archives/
```

给出所有安装的 deb 包的列表命令如下:

```
#dpkg--get-selections \* >| selections.txt
#dpkg--set-selections< selections.txt
```

从包源代码编译新的 deb 包的命令如下(假定源代码目录是 foo-version):

```
#cd foo-version
#su-c "apt-get update ; apt-get install fakeroot"\
#dpkg-buildpackage-rfakeroot-us-uc
#su-c "dpkg-i ../foo_version-revision_arch.deb"
```

17.3 Untangle 系统文件

除了基本 Debian Linux 系统外,Untangle 提供了 bunnicula 目录,管理所有其安全组件配置文件信息、运行状况、日志、数据和 Web 管理接口的层次目录,如下所示。

```
uvm /usr/bin/uvm
bunniculaHome=/usr/share/untangle
bunnicula.lib.dir=/usr/share/untangle/lib(uvm libraries uvmcore jars.)
bunnicula.toolbox.dir=/usr/share/untangle/toolbox(filter or service node jars.)
```

bunnicula.log.dir= /usr/share/untangle/log(log files.)
bunnicula.data.dir= /usr/share/untangle/data(data files.)
bunnicula.db.dir= /usr/share/untangle/db(database files.)
bunnicula.web.dir= /usr/share/untangle/web(servlet directories.)
bunnicula.conf.dir= /usr/share/untangle/conf(configuration files,added to classpath.)
bunnicula.tmp.dir= /usr/share/untangle/tmp(temporary files.)
bunnicula.lang.dir= /usr/share/untangle/lang(languages resources filesI18n.)
bunnicula.skins.dir= ? ? (skins files.)

静态配置文件目录主要在：

/usr/share/untangle/
/usr/share/untangle-apache2-config/
/usr/share/untangle-cli-client/
/usr/share/untangle-cli-server/
/usr/share/untangle-fuzzyocr/
/usr/share/untangle-kiosk/
/usr/share/untangle-ldap-server/
/usr/share/untangle-linux-config/
/usr/share/untangle-net-alpaca/
/usr/share/untangle-nsis-addons/
/usr/share/untangle-shield/
/usr/share/untangle-snort-rules/
/usr/share/untangle-spamassassin-update/
/usr/share/untangle-system-config/
/usr/share/untangle-ujac2pdf/
/usr/share/untangle-update-manager/
/usr/share/untangle-webfilter-init/

本地管理桌面工具集及对应的脚本如下：

/usr/share/untangle-kiosk/homes/kiosk/utils/ut-desktop.sh
/usr/share/untangle-kiosk/homes/kiosk/utils/ut-reboot.sh
/usr/share/untangle-kiosk/homes/kiosk/utils/ut-restore.sh
/usr/share/untangle-kiosk/homes/kiosk/utils/ut-shell.sh
/usr/share/untangle-kiosk/homes/kiosk/utils/screensaver.sh
/usr/share/untangle-kiosk/homes/kiosk/utils/ut-off.sh
/usr/share/untangle-kiosk/homes/kiosk/utils/ut-resolution.sh
/usr/share/untangle/shield
http://localhost:3001

动态配置修改后的脚本文件主要在：

/etc/untangle/
/etc/untangle-ldap/
/etc/untangle-net-alpaca/
/etc/untangle-net-alpaca/tc-rules.d/ 流量管理动态规则生成
/etc/untangle-net-alpaca/iptables-rules.d iptables 动态规则生成

启动 Daemon 与启动配置的脚本在目录/etc/init.d/：

```
untangle-hardware-config
untangle-net-alpaca
untangle-net-alpaca-iptables
untangle-reports
untangle-restore-tools
untangle-shield
untangle-slapd
untangle-vm
watchdog(硬件看门狗)
bypass(软件 bypass)
```

17.4 Untangle 虚拟机

Untangle 系统软件以 Java 代码为主体，这部分代码主要在 uvm-lib 目录中体现，入口点如下：

```
src\uvm-lib\bootstrap\com\untangle\uvm\engine\Main.java
```

Untangle Server 使用虚拟机的概念，称为 Untangle Virtual Machine(UVM)，它是基于 Java 虚拟机(JVM)的一个 Linux 进程，是运行各安全组件(线程)的平台。在 UVM 上，所有安全功能组件都被抽象为 node、agent、vectron。这种实现方法的优势是借助 Java 与平台无关的特性，能够很方便地移植到其他操作系统平台上。另外，所有其他安全组件的抽象可以动态加载、启动、更新、关闭和卸载。

UVM 之上的功能组件提供的各种服务，包括各种管理器、管道等的接口和实现，如配置管理器、备份管理等，应用组件的基类和接口等。

Untangle 借助了很多开源的 Java 类库(\src\uvm-lib\lib*.jar 文件)，如 PostgreSQL JDBC Driver、C3P0、Hibernate、Apache Commons、log4j 等，详见附录 A。

17.5 Linux-UVM 结合部

Linux-UVM 结合部主要是用 C 语言编写的 3 个库 libnetcap、libmvutil、libvector 和 Java JNI 的底层 jnetcap.c、jmvutil.c、jvector.c。其中考虑到效率问题，这部分采用 JNI(Java Native Interface)，Java 代码可以直接调用 libnetcap、libmvutil、libvector 库函数。

底层 libnetcap 采用 pthread 多线程(POSIX Thread)。用户线程进行系统调用发生阻塞时，整个进程都会被阻塞；内核线程允许一个线程被阻塞时，其余正常运行。

17.5.1 实现关键技术

1. 多线程技术

多线程的核心是并发执行多任务，并行(Parallel)与并发(Concurrent)是两个不同的概念。多线程并行执行是指这些活动的线程在不同的硬件资源或者处理单元上同时执行，即

在任何时间点上都是同时执行的。而多个线程并发执行是指在同一个硬件资源上交替执行，即在某个时间点上只有一个线程执行。

线程定义为 CPU 资源占用的一个单位，包括指向指令流中当期指令的程序计数器、当前线程的 CPU 状态信息以及堆栈等其他资源。

实际处理器由大量的资源组成，包括体系结构状态，如寄存器、Cache、总线、执行单元等。通过复制体系结构的状态信息的方法就可以创建多个逻辑处理器（或者线程）。执行资源就可以被不同逻辑处理器共享，这种技术叫做 SMT（Simultaneous Multi-Threading）。与 SMT 对应的是 CMT（Chip Multi-Threading），基于多核 CPU，能够支持真正的硬件多线程技术。

Intel SMT 实现的超线程技术（Hyper-Threading，HT）是通过时延隐藏的方法来提高处理器的性能。超线程上的线程执行并非真正意义上的并行。

进程是离散的程序任务的集合。每个进程拥有独立的地址空间。一个进程的所有线程共享同一个地址空间。每个程序都由一个或多个进程组成，同时每个进程包含一个或多个线程，每个线程都被操作系统调度程序映射到处理器上执行。处理器允许程序员向操作系统请求将特定的线程映射到特定的处理器上。

线程从软件上讲就是相关指令的离散序列。每个程序至少包含一个线程，那就是主线程。主线程负责程序的初始化工作，并且执行初始化指令，随后主线程会为执行各种不同任务而分别创建其他的线程。每个线程都会维持自己当前的机器状态。

从硬件上讲，线程就是一条与其他硬件线程执行路径相互独立的执行路径。操作系统的工作就是将软件线程映射到硬件执行资源上。

线程包括用户级（或应用程序）线程、内核级线程和硬件线程。单个程序线程包含以上三个层面，被操作系统作为内核线程实现，进而作为硬件线程来执行。

操作系统之上的线程的运行原理如下：在线程定义和准备阶段，程序设计环境完成对线程的指定，编译器完成对线程的编译。在运行阶段，操作系统完成对线程的创建和管理。最后，在执行阶段，处理器对线程指令序列进行实际执行。

依赖于运行时环境的应用程序代码称为托管代码（Managed Code）。托管代码在托管环境中运行，托管环境执行应用程序函数并将这些函数转化为对底层操作系统的调用。托管环境包括 JVM 和 CLR。托管环境本身不提供任何调度功能，而是依赖于操作系统的调度。

应用程序线程可以由内建 API 调用来实现，最常用的是显式线程库 pthreads 和 windows 线程库。操作系统内核维护着大量用于追踪进程与线程的表格。绝大多数的线程级行为依赖于内核级线程，如 pthreads 库。内核线程能够提供更好的性能，并且同一进程的多个内核线程能够同时在不同的处理器或执行核上运行。但是内核级线程的相关开销要比用户级线程要高很多。因此经常采取重用内核级线程的策略来降低开销。

1）pthread 基础

（1）线程简介。

在传统 UNIX 编程模型中，当一个进程需要由另一个实体执行某件事时，该进程派生（Fork）一个子进程，让子进程去进行处理。主要的使用场景是 UNIX 下的网络服务器程序，即父进程接受连接，派生子进程，子进程处理与客户的交互。

虽然这种模型使用得较好，但是派生时存在一些问题。

① 派生操作代价比较高。内存映像要从父进程复制到子进程，所有描述符要在子进程中复制等。目前有的 UNIX 实现使用一种叫做写时复制（Copy-on-Write）的技术，可避免父进程数据空间向子进程的复制。尽管有这种优化技术，派生代价依然比较高。

② 派生子进程后，需要用进程间通信（IPC）在父子进程之间传递信息。派生之前的信息容易传递，因为子进程从一开始就有父进程数据空间及所有描述字的副本。但是从子进程返回信息给父进程需要做更多的工作。

线程有助于解决这两个问题。线程有时被称为轻量级进程（Lightweight Process），因为线程比进程“轻量级”，一般来说，创建一个线程要比创建一个进程快 10～100 倍。

一个进程中的所有线程不仅共享全局变量，而且共享进程指令、大多数数据、打开的文件（如描述字）、信号处理程序和信号处置、当前工作目录、用户 ID 和组 ID。但是每个线程有自己的线程 ID、寄存器集合（包括程序计数器和栈指针）、栈（用于存放局部变量和返回地址）、错误（Error）、信号掩码（Sigmask）和优先级。

一个进程中的所有线程共享相同的全局内存，这使得线程很容易共享信息，但是这种简易性也带来了同步问题。

（2）pthread 线程基本函数。

在 Linux 中线程编程符合 POSIX.1 标准，称为 pthreads。所有的 pthread 函数都以 pthread_开头。pthread 库主要有 5 个基本线程函数，在调用前均要包含 pthread.h 头文件。

第一个函数如下：

```
int pthread_create(pthread_t * tid,const pthread_attr_t * attr,void * (* func)(void * ),void * arg);
```

一个进程中的每个线程都由一个线程 ID（thread ID）标识，其数据类型是 pthread_t（常常是 unsigned int）。如果新的线程创建成功，其 ID 将通过 tid 指针返回。

每个线程都有很多属性，其中包括优先级、起始栈大小、是否应该是一个守护线程等，当创建线程时，可通过初始化一个 pthread_attr_t 变量说明这些属性以覆盖默认值。人们通常使用默认值，在这种情况下，将 attr 参数说明为空指针。

最后，当创建一个线程时，需要说明一个它将执行的函数。线程以调用该函数开始，然后或者显式地终止（调用 pthread_exit）或者隐式地终止（让该函数返回）。函数的地址由 func 参数指定，该函数的调用参数是一个指针 arg，如果需要多个调用参数，必须将它们打包成一个结构，然后将其地址当作唯一的参数传递给起始函数。

在 func 和 arg 的声明中，func 函数取一个通用指针（void *）参数，并返回一个通用指针（void *），这就使得人们可以传递一个指针（指向任何想要指向的东西）给线程，由线程返回一个指针（同样指向任何想要指向的东西）。调用成功，返回 0，出错时返回正值。Pthread 函数不设置错误号 errno。

第二个函数如下：

```
int pthread_join(pthread_t tid,void * * status);
```

该函数等待一个线程终止。把线程和进程相比，pthread_creat 类似于 fork，而 pthread_join 类似于 waitpid。人们需要等待线程的 tid，很可惜，人们没有办法等待任意一个线程结束。如果 status 指针非空，线程的返回值（一个指向某个对象的指针）将存放在

status 指向的位置。

第三个函数如下：

```
pthread_t pthread_self(void);
```

每个线程都有一个 ID 以在给定的进程内标识自己。线程 ID 由 pthread_creat 返回，可以调用 pthread_self 取得自己的线程 ID。

第四个函数如下：

```
int pthread_detach(pthread_t tid);
```

线程或者是可汇合的(Joinable)或者是脱离的(Detached)。当可汇合的线程终止时，其线程 ID 和退出状态将保留，直到另外一个线程调用 pthread_join。脱离的线程则像守护进程：当它终止时，所有的资源都释放，人们不能等待它终止。如果一个线程需要知道另一个线程什么时候终止，最好保留第二个线程的可汇合性。pthread_detach 函数将指定的线程变为脱离的。该函数通常被想脱离自己的线程调用，如 pthread_detach(pthread_self())。

第五个函数如下：

```
void pthread_exit(void * status);
```

该函数终止线程。如果线程未脱离，其线程 ID 和退出状态将一直保留到调用进程中的某个其他线程调用 pthread_join 函数。指针 status 不能指向局部于调用线程的对象，因为线程终止时这些对象也消失。有两种其他方法可使线程终止。

① 启动线程的函数(pthread_creat 的第 3 个参数)返回。既然该函数必须说明为返回一个 void 指针，该返回值便是线程的终止状态。

② 如果进程的 main 函数返回或者任何线程调用了 exit，进程将终止，线程将随之终止。

(3) pthread 线程属性设置。

① 线程属性设置。

用 pthread_create 函数创建一个线程，在这个线程中，使用默认参数，即将该函数的第二个参数设为 NULL。的确，对大多数程序来说，使用默认属性就足够了，但还是有必要来了解一下线程的有关属性。

属性结构为 pthread_attr_t，它同样在头文件 pthread.h 中定义，属性值不能直接设置，必须使用相关函数进行操作，初始化的函数为 pthread_attr_init，这个函数必须在 pthread_create 函数之前调用。属性对象主要包括是否绑定、是否分离、堆栈地址、堆栈大小、优先级。默认的属性为非绑定、非分离、缺省的堆栈、与父进程同样级别的优先级。

② 绑定。

关于线程的绑定，牵涉另外一个概念：轻量级进程(Light Weight Process，LWP)。轻量级进程可以理解为内核线程，它位于用户层和系统层之间。系统对线程资源的分配、对线程的控制是通过轻量级进程来实现的，一个轻量级进程可以控制一个或多个线程。默认情况下，启动多少轻量级进程、哪些轻量级进程来控制哪些线程是由系统来控制的，这种状况称为非绑定的。绑定状况下，某个线程固定地绑在一个轻量级进程之上。被绑定的线程具有较高的响应速度，这是因为 CPU 时间片的调度是面向轻量级进程的，绑定的线程可以保

证在需要的时候它总有一个轻量级进程可用。通过设置被绑定的轻量级进程的优先级和调度级可以使得绑定的线程满足诸如实时反应之类的要求。

设置线程绑定状态的函数为 pthread_attr_setscope，它有两个参数，第一个是指向属性结构的指针；第二个是绑定类型，它有两个取值：PTHREAD_SCOPE_SYSTEM（绑定的）和 PTHREAD_SCOPE_PROCESS（非绑定的）。下面的代码即创建了一个绑定的线程：

```
pthread_attr_t attr;
pthread_t tid;
```

属性值初始化时，均设为默认值。

```
pthread_attr_init(&attr);
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
pthread_create(&tid, &attr,(void * )my_function, NULL);
```

③ 线程的分离状态。

线程的分离状态决定一个线程以什么样的方式来终止自己。非分离的线程终止时，其线程 ID 和退出状态将保留，直到另外一个线程调用 pthread_join。分离的线程在当它终止时，所有的资源将释放，人们不能等待它终止。设置线程分离状态的函数为 pthread_attr_setdetachstate(pthread_attr_t * attr，int detachstate)。第二个参数可选为 PTHREAD_CREATE_DETACHED（分离线程）和 PTHREAD _CREATE_JOINABLE（非分离线程）。这里要注意的一点是，如果设置一个线程为分离线程，而这个线程运行又非常快，它很可能在 pthread_create 函数返回之前就终止了，它终止以后就可能将线程号和系统资源移交给其他的线程使用，这样调用 pthread_create 的线程就得到了错误的线程号。要避免这种情况可以采取一定的同步措施，最简单的方法之一是可以在被创建的线程里调用 pthread_cond_timewait 函数，让这个线程等待一会儿，留出足够的时间让函数 pthread_create 返回。设置一段等待时间是在多线程编程里常用的方法。

④ 优先级。

它存放在结构 sched_param 中。用函数 pthread_attr_getschedparam 和函数 pthread_attr_setschedparam 进行存放，一般说来，人们总是先取优先级，对取得的值修改后再存放回去。下面即是一段简单的例子：

```
pthread_attr_t attr; pthread_t tid;
sched_param param;
int newprio=20;
```

属性值初始化时，均设为默认值。

```
pthread_attr_init(&attr);
```

将优先级值传入 pthread 属性值。

```
pthread_attr_getschedparam(&attr, &param);
param.sched_priority=newprio;
pthread_attr_setschedparam(&attr, &param);
pthread_create(&tid, &attr,(void * )myfunction, myarg);
```

`(4) 创建线程特定数据的键值。

单线程 C 程序有两类基本数据：局部数据(Local)和全局数据(Global)。

对于多线程 C 程序，添加了第三类数据：线程特定数据或特定于线程的数据(TSD/TLS，Thread Local Data/State Data)。线程特定数据与全局数据非常相似，区别在于前者为线程专有。

线程特定数据由每个线程进行维护，每个线程特定数据项都与一个作用于进程内所有线程的键(key)关联。通过使用 key，线程可以访问基于每个线程进行维护的指针(void *)。

pthread_key_create 语法如下：

```
int    pthread_key_create(pthread_key_t * key, void(* destructor)(void * ));
```

pthread_key_create() 在成功完成之后返回零。其他任何返回值都表示出现了错误。如果出现以下任一情况，pthread_key_create() 将失败并返回相应的值。

① EAGAIN。

描述：key 名称空间已经用完。

② ENOMEM。

描述：此进程中虚拟内存不足，无法创建新键。

(5) 删除线程特定数据的键值。

使用 pthread_key_delete() 可以销毁现有线程特定数据键。由于键已经无效，因此将释放与该键关联的所有内存。引用无效键将返回错误。Solaris 线程中没有类似的函数。

pthread_key_delete 语法如下：

```
int    pthread_key_delete(pthread_key_t key);
```

pthread_key_delete() 在成功完成之后返回零。其他任何返回值都表示出现了错误。如果出现以下情况，pthread_key_delete() 将失败并返回相应的值。

EINVAL

描述：key 的值无效。

(6) 设置线程特定数据

使用 pthread_setspecific() 可以为指定线程特定数据键设置线程特定绑定。

pthread_setspecific 语法如下：

```
int    pthread_setspecific(pthread_key_t key, const void * value);
```

pthread_setspecific() 在成功完成之后返回零。其他任何返回值都表示出现了错误。如果出现以下任一情况，pthread_setspecific() 将失败并返回相应的值。

① ENOMEM。

描述：虚拟内存不足。

② EINVAL。

描述：key 无效。

注意：设置新绑定时，pthread_setspecific() 不会释放其存储空间。必须释放现有绑定，否则会出现内存泄漏。

(7) 获取线程特定数据。

使用 pthread_getspecific() 获取调用线程的键绑定,并将该绑定存储在 value 指向的位置中。

pthread_getspecific 语法如下

```
void    * pthread_getspecific(pthread_key_t key);
```

pthread_getspecific 不返回任何错误。

和进程相比,线程的最大优点之一是数据的共享性,各个进程共享父进程的数据段,可以方便读取、修改数据。但这也给多线程编程带来了许多问题。必须当心有多个不同的进程访问相同的变量。许多函数是不可重入的,即同时不能运行一个函数的多个副本(除非使用不同的数据段)。在函数中声明的静态变量常常带来问题,函数的返回值也会有问题。因为如果返回的是函数内部静态声明的空间的地址,则在一个线程调用该函数得到地址后使用该地址指向的数据时,别的线程可能调用此函数并修改了这一段数据。在进程中共享的变量必须用关键字 volatile 来定义,这是为了防止编译器在优化时(如 gcc 中使用-ox 参数)改变它们的使用方式。为了保护变量,必须使用信号量、互斥等方法来保证对变量的正确使用。

2) pthread 详解

(1) 线程模型。

从 Linux 2.6 内核开始,Linux 采用了 NPTL(Native POSIX Thread Library)线程模型。NPTL 是一个 1×1 线程函数库。用户产生的线程与内核能够分配的对象之间的联系是一对一的,这是所有线程程序中最简单的。

pthread 由 Thread ID、Stack、Policy、Signal mask、Errno、Thread-Specific Data 组成。

(2) 线程标识。

① pthread_t 用于表示 Thread ID,具体内容根据实现的不同而不同,有可能是一个 Structure,因此不能将其看做整数。

② pthread_equal 函数用于比较两个 pthread_t 是否相等,格式如下:

```
#includeint pthread_equal(pthread_t tid1, pthread_t tid2)
```

③ pthread_self 函数用于获得本线程的 Thread ID,格式如下:

```
#includethread _t pthread_self(void);
```

(3) 创建线程。

① 创建线程可以调用 pthread_create 函数:

```
int pthread_create(
pthread_t * restrict tidp,                   \//返回最后创建出来的线程的 Thread ID
const pthread_attr_t * restrict attr,        \//指定线程的属性,现在可以用 NULL
void * (* start_rtn)(void * ),              \//指定线程函数指针,该函数返回一个 void,参数也为 void
void * restrict arg);                        \//传入给线程函数的参数
```

② pthread 函数在出错的时候不会设置 errno,而是直接返回错误值。

③ 在 Linux 系统下面,在 2.6 内核之前,线程被看作一种特殊、可共享地址空间和资源的进程,因此在同一个进程中创建的不同线程具有不同的 Process ID(调用 getpid 函数获

得）。Linux 采用了 NPTL（Native POSIX Thread Library）线程模型。在该线程模型下同一进程下不同线程调用 getpid 返回同一个 Process ID。

④ 不能对创建的新线程和当前创建者线程的运行顺序做出任何假设。

（4）线程终结。

① exit、_Exit、_exit 用于中止当前进程，而非线程。

② 中止线程可以有三种方式。

a. 在线程函数中返回。

b. 被同一进程中的另外的线程取消掉。

c. 线程调用 pthread_exit 函数。

③ pthread_exit 和 pthread_join 函数的用法。

a. 线程 A 调用 pthread_join(B,&rval_ptr)，被阻塞，进入分离状态（如果已经进入分离状态，则 pthread_join 函数返回 EINVAL）。如果对 B 的结束代码不感兴趣，则 rval_ptr 可以传 NULL。

b. 线程 B 调用 pthread_exit(rval_ptr)，退出线程 B，结束代码为 rval_ptr。注意 rval_ptr 指向内存的生命周期，不应该指向 B 的堆栈中的数据。

c. 线程 A 恢复运行，pthread_join 函数调用结束，线程 B 的结束代码被保存到 rval_ptr 参数中去。如果线程 B 被取消，那么 rval_ptr 的值就是 PTHREAD_CANCELLED。

两个函数原型如下：

```
void pthread_exit(void * rval_ptr);
int pthread_join(pthread_t thread, void **rval_ptr);
```

④ 一个线程可以要求另一个线程被取消，通过调用 pthread_cancel 函数：

```
void pthread_cancel(pthread_t tid);
```

该函数会使指定线程如同调用了 pthread_exit(PTHREAD_CANCELLED)。不过，指定线程可以选择忽略或者进行自己的处理，在后面会讲。此外，该函数不会导致 Block，只是发送 Cancel 这个请求。

⑤ 线程可以安排在它退出的时候，某些函数自动被调用，类似 atexit 函数。需要调用如下函数：

```
void pthread_cleanup_push(void(* rtn)(void * ), void * arg);
void pthread_cleanup_pop(int execute);
```

这两个函数维护一个函数指针的 Stack，可以把函数指针和函数参数值 push/pop。执行的顺序则是从栈顶到栈底，也就是和 push 的顺序相反。

在下面情况下，pthread_cleanup_push 所指定的 thread cleanup handlers 会被调用：

a. 调用 pthread_exit。

b. 相应 Cancel 请求。

c. 以非 0 参数调用 pthread_cleanup_pop()。如果以 0 调用 pthread_cleanup_pop()，那么 handler 不会被调用。

由于这两个函数可能用宏的方式来实现，因此这两个函数的调用必须得是在同一个

Scope 之中，并且配对，因为在 pthread_cleanup_push 的实现中可能有一个“{”，而 pthread_cleanup_pop 可能有一个“}”。因此，一般情况下，这两个函数是用于处理意外情况用的，举例如下：

```
void * thread_func(void * arg)
{
pthread_cleanup_push(cleanup, "handler")
// do something
Pthread_cleanup_pop(0);
return((void * )0);
}
```

⑥ 默认情况下，一个线程 A 的结束状态被保存下来直到 pthread_join 为该线程被调用过，也就是说即使线程 A 已经结束，只要没有线程 B 调用 pthread_join(A)，A 的退出状态则一直被保存。而当线程处于 Detached 状态之时，当线程退出的时候，其资源可以立刻被回收，那么这个退出状态也丢失了。在这个状态下，无法为该线程调用 pthread_join 函数。我们可以通过调用 pthread_detach 函数来使指定线程进入 Detached 状态：

```
int pthread_detach(pthread_t tid);
```

通过修改调用 pthread_create 函数的 attr 参数，可以指定一个线程在创建之后立刻就进入 Detached 状态。

⑦ 进程函数和线程函数的相关性如表 17-1 所示。

表 17-1　进程函数和线程函数的相关性

原语进程	原 语 线 程	描　　述
fork	pthread_create	创建新的控制流
exit	pthread_exit	退出已有的控制流
waitpid	pthread_join	等待控制流并获得结束代码
atexit	pthread_cleanup_push	注册在控制流退出时候被调用的函数
getpid	pthread_self	获得控制流的 ID
abort	pthread_cancel	请求非正常退出

(5) 线程同步。

① 互斥量：Mutex。

a. 用于互斥访问。

b. 类型：pthread_mutex_t，必须被初始化为 PTHREAD_MUTEX_INITIALIZER(用于静态分配的 mutex，等价于 pthread_mutex_init(…，NULL))或者调用 pthread_mutex_init 函数。Mutex 也应该用 pthread_mutex_destroy 函数来销毁。这两个函数原型如下：

```
#include
int pthread_mutex_init(
pthread_mutex_t * restrict mutex,
```

```
const pthread_mutexattr_t * restrict attr)
int pthread_mutex_destroy(pthread_mutex_t * mutex);
```

c. pthread_mutex_lock 函数用于 Lock Mutex，如果 Mutex 已经被 Lock，该函数调用会 Block 直到 Mutex 被 Unlock，然后该函数会 Lock Mutex 并返回。与 pthread_mutex_trylock 类似，只是当 Mutex 被锁死的时候不会被阻塞，而是返回一个错误值 EBUSY。pthread_mutex_unlock 函数则是 unlock 一个 mutex。这三个函数原型如下：

```
int pthread_mutex_lock(pthread_mutex_t * mutex);
int pthread_mutex_trylock(pthread_mutex_t * mutex);
int pthread_mutex_unlock(pthread_mutex_t * mutex);
```

② 读写锁：Reader-Writer Locks。

a. 多个线程可以同时获得读锁，但是只有一个线程能够获得写锁。

b. 读写锁有如下三种状态。

a）一个或者多个线程获得读锁，其他线程无法获得写锁。

b）一个线程获得写锁，其他线程无法获得读锁。

c）没有线程获得此读写锁。

c. 类型为 pthread_rwlock_t。

d. 创建和关闭读写锁的方法如下：

```
int pthread_rwlock_init(
pthread_rwlock_t * restrict rwlock,
const pthread_rwlockattr_t * restrict attr)
int pthread_rwlock_destroy(pthread_rwlock_t * rwlock);
```

e. 获得读写锁的方法如下：

```
int pthread_rwlock_rdlock(pthread_rwlock_t * rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t * rwlock);
int pthread_rwlock_unlock(pthread_rwlock_t * rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t * rwlock);
int pthread_rwlock_trywrlock(pthread_rwlock_t * rwlock);
```

pthread_rwlock_rdlock：获得读锁。

pthread_rwlock_wrlock：获得写锁。

pthread_rwlock_unlock：释放锁，不管是读锁还是写锁都是调用此函数。

注意：具体实现可能对同时获得读锁的线程个数有限制，所以在调用 pthread_rwlock_rdlock 的时候需要检查错误值，而另外两个 pthread_rwlock_wrlock 和 pthread_rwlock_unlock 则一般不用检查。

③ Conditional Variable：条件变量

a. 条件必须被 Mutex 保护起来。

b. 类型为 pthread_cond_t，必须被初始化为 PTHREAD_COND_INITIALIZER（用于静态分配的条件，等价于 pthread_cond_init（…，NULL））或者调用 pthread_cond_init。

```
int pthread_cond_init(
```

```
pthread_cond_t * restrict cond,
const pthread_condxattr_t * restrict attr)
int pthread_cond_destroy(pthread_cond_t * cond);
```

c. pthread_cond_wait 函数用于等待条件发生(=true)。pthread_cond_timedwait 类似,只是当等待超时的时候返回一个错误值 ETIMEDOUT。超时的时间用 timespec 结构指定。此外,两个函数都需要传入一个 Mutex 用于保护条件,函数如下:

```
int pthread_cond_wait(
pthread_cond_t * restrict cond,
pthread_mutex_t * restrict mutex);
int pthread_cond_timedwait(
pthread_cond_t * restrict cond,
pthread_mutex_t * restrict mutex,
const struct timespec * restrict timeout);
```

d. timespec 结构定义如下:

```
struct timespec {
time_t tv_sec;                    /* seconds * /
long tv_nsec;                     /* nanoseconds * /
};
```

注意:timespec 的时间是绝对时间而非相对时间,因此需要先调用 gettimeofday 函数获得当前时间,再转换成 timespec 结构,加上偏移量。

e. 有两个函数用于通知线程条件被满足(=true):

```
int pthread_cond_signal(pthread_cond_t * cond);
int pthread_cond_broadcast(pthread_cond_t * cond);
```

两者的区别是前者会唤醒单个线程,而后者会唤醒多个线程。

3) Java 多线程

JVM 充分利用了多线程技术(Multiple Thread),它至少创建了三个线程:执行多线程、垃圾回收线程和用于即时编译执行技术 JIT 的编译线程。

Java 通过 java. lang. Thread 类来实现多线程(通过 Runnable 接口实现多线程,只是一种实现方式,这种方式还是需要借助 java. lang. Thread 类,才能启动新的线程)。

2. Java Native Invocation 技术

Java 语言是解释型语言,除非采用了即时编译技术,通常情况下 JIT 最终编译后的程序是不能作为机器代码直接执行,需要 Java 虚拟机的解释执行。通过 Java 本地接口(Java Native Interface,JNI)技术,把计算量大的任务转移给本地方法,可以提高处理效率和系统性能。通过 JNI 技术架构实现 C、C++ 与 Java 应用的双向调用。

JNI 技术主要用法如下:

```
class Test {
    public native void func();
    static {
```

```
        System.loadLibrary("Test");
        }

        Public static void main(String[] args){
        ⋮
        }
}

javac Test.java                        //生成 Test.class

javah Test                             //生成 Test.h
```

在 Test.h 中，这个本地方法声明之前有关键词 JNIEXPORT，接着是这个方法的返回值与关键字 JNICALL，然后是这个方法的名称，名称由 Java_前缀与 Java 类名加“_”和方法名称组成，最后在方法的参数中包括 JNIEnv * 与 jobject 参数。

JNIEnv 是 JNIEnv 接口的指针，通过该指针才能将 Java 应用中要访问的参数或对象传递给 C 语言段的应用。其中 jobject 是当前对象本身的引用，相当于 Java 应用中的 this 变量。

在 C 语言端代码主文件如下：

```
#include<jni.h>
#include "Test.h"
⋮
JNIEXPORT void JNICALL Java_Test_func(JNIEnv * env, jobject obj)
{
⋮
}
```

在 Linux 中编译成.so 文件，语句如下：

```
gcc-shared-I$JDK_HOME/include-I/usr/include * .c-o libtest.so
```

将 lib *.so 文件放入系统变量 LD_LIBRARY_PATH：

```
#setenv  /* .so 目录/:$LD_LIBRARY_PATH
```

JVM 中要调用以上库函数，必须加载，语句如下：

```
System.loadLibrary("uvmcore");
```

由于语言环境的信息交换，存在数据类型的对应问题。语言间数据类型的互相转换的过程。如 Java 应用中调用 C 语言的方法，传递了一个 int 整型，JVM 根据 native 关键词转换为本地类型传递给本地应用。具体参见 JNI 基本数据类型与本地数据类型的对照。

3. I/O 多路复用机制

1）I/O 模型

I/O 模型主要有五种：阻塞 I/O 模型、非阻塞 I/O 模型、I/O 复用(select 和 poll)模型、信号驱动 I/O(SIGIO)模型和异步 I/O 模型。

(1) 阻塞 I/O 模型。

进程调用 recvfrom()系统调用直到数据到达且复制到应用层缓冲区或是出错才返回。常见错误有系统调用被信号中断,进程阻塞的时间是指从调用 recvfrom()开始到它返回的这段时间,当进程返回成功指示时,应用进程开始处理数据。

(2) 非阻塞 I/O 模型。

当请求的 I/O 操作不能完成时,进程不会阻塞,会返回一个错误。如前三次调用 recvfrom()时仍无数据返回,内核返回一个错误。第四次调用 recvfrom()时,数据已准备好,被复制到应用缓冲区,recvfrom()返回成功指示,接着处理数据。由于采用轮询方法,对 CPU 负载要求比较高。

(3) I/O 复用模型。

调用 select 或 poll,在这两个系统调用中的某一个上阻塞,而不是阻塞于真正 I/O 系统调用。阻塞于 select 调用,等待数据报套接口可读。当 select 返回套接口可读条件时,调用 recevfrom()将数据报复制到应用缓冲区中。

(4) 信号驱动 I/O 模型。

启动信号驱动 I/O 调用时,先通过系统调用 sigaction 注册一个信号处理程序。此系统调用立即返回,进程继续工作,它是非阻塞的。当数据准备好被读时,就为该进程生成一个 SIGIO 信号。随即在信号处理程序中调用 recvfrom()来读数据,并通知主循环数据已准备好被处理中。也可以通知主循环,让它来读数据。

(5) 异步 I/O 模型。

让内核启动操作,并在整个操作完成后(包括将数据从内核复制到用户自己的缓冲区)通知用户。

当从一个文件描述符读,然后写到另一个文件描述符,可以在下列形式的循环中使用阻塞 I/O。

```
while((n=read(STDIN_FILENO,buf,BUFSIZ))>0)
    if(write(STDOUT_FILENO,buf,n)!=n)
        err_sys("write error");
```

但是,如果必须从两个描述符读,那么就可能长时间阻塞在一个描述符上,而另一个描述符虽然有很多数据,却不能得到及时处理。

2) I/O 多路复用实例

I/O 多路复用(I/O Multiplexing),先构造一张有关描述符的列表,然后调用一个函数,直到这些描述符中的一个已准备好进行 I/O 时,该函数才返回。在返回时,它告诉进程哪些描述符已准备好可以进行 I/O。POSIX.1 标准定义了 select 函数和 pselect 函数,而 poll 函数则是对该基本部分的 XSI 扩展。

(1) select 函数。

select 函数如下:

```
#include<sys/select.h>


int select(int maxfdp1,fd_set*restrict readfds,fd_set*restrict writefds,fd_set*restrict exceptfds,
```

```
struct timeval * restrict tvptr);Returns:count of ready descriptors,0 on timeout,-1 on error
```

最后一个参数 tvptr 指定自愿等待的时间。

tvptr==NULL,永久等待。当所指定的描述符中的一个已经准备好或捕捉到一个信号则返回。如果捕捉到一个信号,则 select 返回-1,errno 设置为 EINTR。

tvptr->tv_sec==0 && tvptr->tv_usec==0,完全不等待。这是得到多个描述符状态而不阻塞 select 函数的轮询方法。

tvptr->tv_sec! =0 || tvptr->tv_usec! =0,等待指定的秒数和微秒数。当指定的描述符之一已准备好,或当指定的时间值已经超时,或捕捉到信号时,函数返回。在 Linux 下,若在该时间值尚未超过时 select 就返回,那么将用余留时间更新 tvptr 指向的结构。

中间三个参数 readfds、writefds 和 exceptfds 是指向描述符集的指针。这三个描述符集说明了相关的可读、可写或出现异常条件的各个描述符。每个描述符集存放在一个 fd_set 数据类型中。这种数据类型为每一可能的描述符保持一位。描述符集的函数接口(可能实现为宏)包括:调用 FD_ZERO 将一个指定的 fd_set 变量的所有位设置为 0;调用 FD_SET 设置一个 fd_set 变量的指定位;调用 FD_CLR 将一指定位清除;调用 FD_ISSET 测试一指定位是否设置。描述符集的函数接口如下:

```
#include<sys/select.h>
int FD_ISSET(int fd,fd_set * fdset);
    Returns:nonzero if fd is in set,0 otherwise
void FD_CLR(int fd,fd_set * fdset);
void FD_SET(int fd,fd_set * fdset);
void FD_ZERO(fd_set * fdset);
```

select 函数的第一个参数 maxfdp1 的意思是最大描述符加 1,即在三个描述符集中找出最大描述符编号值,然后加 1。

select 函数有三个可能的返回值。

① 返回值-1 表示出错,如捕捉到一个信号。在这种情况下,将不修改其中任何描述符集。

② 返回值 0 表示没有描述符准备好。

③ 正返回值表示已经准备好的描述符数,该值是三个描述符集中已经准备好的描述符数之和。

对于准备好的意思是,对读写集中的一个描述符的 read 或 write 操作不会阻塞,对异常状态集中的一个描述符有一个未决异常状态(包括①在网络连接上到达的带外数据②在处于数据包模式的伪终端上发生了某些状态)。如果在一个描述符上碰到了文件结尾处,则 select 认为该描述符是可读的,因为调用 read 将返回 0。

timeval 结构如下:

```
struct timeval{
    long tv_sec;                    //second
    long tv_usec;                   //minisecond
}
```

timeout 设置情况如下。

① null：select 将一直被阻塞，直到某个文件描述符上发生了事件。

② 0：仅检测描述符集合的状态，然后立即返回，并不等待外部事件的发生。

③ 特定的时间值：如果在指定的时间段里没有事件发生，select 将超时返回。文件描述符集 fdset 中的文件描述符的个数是有限制的，最大值由 FD_SETSIZE 指定，在 Linux 中为 1024。

（2）pselect 函数。

pselect 函数如下：

```
#include<sys/select.h>

int pselect(int maxfdp1, fd_set * restrict readfds, fd_set * restrict writefds, fd_set * restrict
exceptfds, const struct timespec * restrict tsptr, const sigset_t * restrict sigmask);
Returns: count of ready descriptors, 0 on timeout, -1 on error
```

它与 select 函数的区别如下。

① pselect 使用 timespec 结构指定超时值。timespec 结构以秒和纳秒表示时间，而非秒和微秒。

② pselect 的超时值被声明为 const，这保证了调用 pselect 不会改变 timespec 结构。

③ pselect 可使用一个可选择的信号屏蔽字。在调用 pselect 时，以原子操作的方式安装该信号屏蔽字，在返回时恢复以前的信号屏蔽字。

（3）poll 函数。

poll 函数提供的功能与 select 函数类似，不过在处理流设备时，它能够提供额外的信息。poll 函数起源于 SVR3，最初局限于流设备。SVR4 取消了这种限制，允许 poll 函数工作在任何描述字上。

poll 函数如下：

```
#include <poll.h>

int poll(struct pollfd fdarray[], nfds_t nfds, int timeout);
Returns: count of ready descriptors, 0 on timeout, 1 on error
struct pollfd {
    int fd;                 /* file descriptor to check, or< 0 to ignore */
    short events;           /* events of interest on fd */
    short revents;          /* events that occurred on fd */
};
```

与 select 函数不同，poll 函数不是为每个状态构造一个描述符集，而是构造一个 pollfd 结构数组，每个数组元素指定一个描述符编号以及对其所关心的状态。pollfd 结构中的 events 告诉内核我们对该描述符关心的是什么。poll 函数返回时，内核设置 revents 成员，以说明对该描述符已经发生了什么事件。fdarray 数组中的元素数由 nfds 说明。

poll 函数可用的测试值如表 17-2 所示。

表 17-2　poll 函数可用的测试值

常　量	说　明	常　量	说　明

POLLIN	普通或优先级带数据可读	POLLWRNORM	普通数据可写
POLLRDNORM	普通数据可读	POLLWRBAND	优先级带数据可写
POLLRDBAND	优先级带数据可读	POLLERR	发生错误
POLLPRI	高优先级数据可读	POLLHUP	发生挂起
POLLOUT	普通数据可写	POLLNVAL	描述字不是一个打开的文件

注意：后三个只能作为描述字的返回结果存储在 revents 中，而不能作为测试条件用于 events 中。最后一个参数 timeout 是指定 poll 函数返回前等待多长时间，它的取值如表 17-3 所示。

表 17-3　timeout 的取值及说明

timeout 值	说　　明	timeout 值	说　　明
INFTIM	永远等待	>0	等待指定数目的毫秒数
0	立即返回，不阻塞进程		

（4）epoll 函数。

epoll 函数是 Linux 下多路复用 I/O 接口 select/poll 的增强版本，它能显著减少程序在大量并发连接中只有少量活跃的情况下的系统 CPU 利用率，因为它不会复用文件描述符集合来传递结果而迫使开发者每次等待事件之前都必须重新准备要被侦听的文件描述符集合，另一个原因就是获取事件的时候，它无须遍历整个被侦听的描述符集，只要遍历那些被内核 I/O 事件异步唤醒而加入 Ready 队列的描述符集合就行了。epoll 除了提供 select/poll 那种 I/O 事件的水平触发（Level Triggered）外，还提供了边沿触发（Edge Triggered），这就使得用户空间程序有可能缓存 I/O 状态，减少 epoll_wait/epoll_pwait 的调用，提高应用程序效率。

epoll 的接口非常简单，一共有三个函数。

① 第一个函数 epoll_create 如下：

```
int epoll_create(int size);
```

该函数创建一个 epoll 的句柄，size 用来告诉内核这个监听的数目一共有多少。这个参数不同于 select 函数中的第一个参数，给出最大监听的 fd+1 的值。需要注意的是，当创建好 epoll 句柄后，它就会占用一个 fd 值，在 Linux 下如果查看/proc/进程 id/fd/，能看到这个 fd，所以在使用完 epoll 后，必须调用 close 关闭，否则可能导致 fd 被耗尽。

② 第二个函数 epoll_ctl 如下：

```
int epoll_ctl(int epfd,int op,int fd,struct epoll_event * event);
```

该函数是 epoll 的事件注册函数，它不同于 select 函数，是在监听事件时告诉内核要监听什么类型的事件，而是在这里先注册要监听的事件类型。

第一个参数是 epoll_create 函数的返回值。

第二个参数表示动作，用三个宏来表示，如表 17-4 所示。

表 17-4 宏及说明

宏	说　　明
EPOLL_CTL_ADD	注册新的 fd 到 epfd 中
EPOLL_CTL_MOD	修改已经注册的 fd 的监听事件
EPOLL_CTL_DEL	从 epfd 中删除一个 fd

第三个参数是需要监听的 fd。

第四个参数是告诉内核需要监听什么事件。

epoll_event 的结构如下：

```
struct epoll_event {
    __uint32_t events;                  /* Epoll events */
    epoll_data_t data;                  /* User data variable */
};
```

events 可以是几个常量的集合，如表 17-5 所示。

表 17-5 常量及说明

常　　量	说　　明
EPOLLIN	对应的文件描述符可以读(包括对端 Socket 正常关闭)
EPOLLOUT	对应的文件描述符可以写
EPOLLPRI	对应的文件描述符有重要优先级的数据可读(这里应该表示有外数据到来)
EPOLLERR	对应的文件描述符发生错误
EPOLLHUP	对应的文件描述符被挂起
EPOLLET	将 EPOLL 设为边缘触发(Edge Triggered)模式
EPOLLLT	将 EPOLL 设为水平触发(Level Triggered)模式
EPOLLONESHOT	只监听一次事件，当监听完这次事件之后，如果还需要继续监听这个 Socket，需要再次把这个 Socket 加入到 EPOLL 队列里

③ 第三个函数 epoll_wait 如下：

```
int epoll_wait(int epfd,struct epoll_event * events,int maxevents,int timeout);
```

等待事件的产生，类似于 select 函数调用。参数 events 用来从内核得到事件的集合，maxevents 告之内核这个 events 有多大，这个 maxevents 的值不能大于创建 epoll_create 函数时的 size，参数 timeout 是超时时间(毫秒，0 会立即返回，－1 将不确定，也有说法是永久阻塞)。该函数返回需要处理的事件数目，如返回 0 表示已超时。

3) epoll 机制

(1) epoll 简介。

epoll 机制是 Linux 2.6 内核引入的 I/O 多路复用的一种方式，用于提高网络 I/O 性能，主要是模拟 AIO 模式(异步 I/O)。

epoll 机制主要是解决 select 函数效率的问题。在 Linux 网络编程中，很长一段时间都

是采用 select 函数来实现多事件触发处理的。select 函数存在如下几个方面的问题。

① 每次调用时要从用户态重复读入参数。

② 每次调用时要重复地扫描文件描述符。

③ 每次调用开始时,要把当前进程放入各个文件描述符的等待队列。调用结束后,又把进程从各个等待队列中删除。

select 采用轮询的方式来处理事件触发,当随着监听 Socket 的文件描述符 fd 的数量增加时,轮询的时间也就越长,造成效率低下。并且在 linux/posix_types.h 中定义了 fd 最大数目是 1024(#define __FD_SETSIZE 1024)。

采用 epoll 机制的优势是支持一个进程打开大数目的 Socket 描述符(fd),不随监听的 fd 数目的增长而降低效率,使用 mmap 加速内核与用户空间的消息传递。

mmap 机制是 Linux 操作系统提供的内核数据映射功能,能把内核态的数据直接映射到用户态。网络处理中需要把数据包从内核态复制到用户态再进行处理。mmap 机制使用户态程序可以直接访问这些内核态数据,而不需要再复制一次,从而节省了开销。

(2) epoll 工作模式。

epoll 机制有两种:ET(Edge Triggered)和 LT 模式(Level Triggered)。ET 是高速模式,只能以非阻塞方式进行,LT 相当于快速的 select,有阻塞和非阻塞两种方式,epoll 通过把操作拆分为 epoll_create、epoll_ctl、epoll_wait 三个步骤避免重复地遍历要监视的文件描述符。

man 手册中给出了 epoll 使用的一个例子。

① 把一个用来从管道中读取数据的文件句柄(RFD)添加到 epoll 描述符。

② 这时从管道的另一端写入 2KB 的数据。

③ 调用 epoll_wait(2),并且它会返回 RFD,说明它已经准备好读取操作。

④ 读取 1KB 的数据。

⑤ 调用 epoll_wait(2)。

……

(3) Edge Triggered 工作模式。

如果在第①步将 RFD 添加到 epoll 描述符的时候使用了 EPOLLET 标志,那么在第⑤步调用 epoll_wait(2)之后将有可能会挂起,因为剩余的数据还存在于文件的输入缓冲区内,而且数据发出端还在等待一个针对已经发出数据的反馈信息。只有在监视的文件句柄上发生了某个事件的时候 ET 工作模式才会汇报事件。因此在第⑤步的时候,调用者可能会放弃等待仍存在于文件输入缓冲区内的剩余数据。在上面的例子中,会有一个事件产生在 RFD 句柄上,因为在第②步执行了一个写操作,然后,事件将会在第③步被销毁。因为第④步的读取操作没有读空文件输入缓冲区内的数据,因此在第⑤步调用 epoll_wait(2)完成后,是否挂起是不确定的。epoll 工作在 ET 模式的时候,必须使用非阻塞套接口,以避免由于一个文件句柄的阻塞读/阻塞写操作把处理多个文件描述符的任务遏死。最好以下面的方式调用 ET 模式的 epoll 接口,在后面会介绍避免可能的缺陷。

① 基于非阻塞文件句柄。

② 只有当 read(2)或者 write(2)返回 EAGAIN 时才需要挂起,等待。但这并不是说每

次 read()时都需要循环读，直到读到产生一个 EAGAIN 才认为此次事件处理完成，当 read()返回读到的数据长度小于请求的数据长度时，就可以确定此时缓冲中已没有数据了，也就可以认为此事读事件已处理完成。

(4) Level Triggered 工作模式。

相反地，以 LT 方式调用 epoll 接口时，它就相当于一个速度比较快的 poll/select，在 poll 能用的地方 epoll 都可以用，因为它们具有同样的职能。即使使用 ET 模式的 epoll，在收到多个数据包的时候仍然会产生多个事件。调用者可以设定 EPOLLONESHOT 标志，在 epoll_wait 收到事件后，epoll 会与事件关联的文件句柄从 epoll 描述符中禁止掉。因此当 EPOLLONESHOT 设定后，使用带有 EPOLL_CTL_MOD 标志的 epoll_ctl 处理文件句柄就成为调用者必须做的事情。

以上是 man 手册对 epoll 中两种模式的简要介绍，这里有必要对两种模式进行详细介绍。

LT 是默认的工作方式，并且同时支持 block 和 no-block Socket，在这种做法中，内核会告诉调用者一个文件描述符是否就绪了，然后调用者可以对这个就绪的 fd 进行 I/O 操作。如果不做任何操作，内核还会继续通知调用者，所以，这种模式编程出错误可能性要小一点。传统的 select/poll 都是这种模型的代表。LT 模式跟 select 有一样的语义，如果可读就触发。例如，某管道原来为空，如果有一个进程写入 2KB 数据，就会触发。如果处理进程读取 1KB 数据，下次轮询时继续触发。该模式下，默认不可读，只有 epoll 通知可读才是可读，否则不可读。

ET 是高速工作方式，只支持 no-block Socket。在这种模式下，当描述符从未就绪变为就绪时，内核通过 epoll 告诉调用者，然后它会假设调用者知道文件描述符已经就绪，并且不会再为那个文件描述符发送更多的就绪通知，直到调用者做了某些操作导致那个文件描述符不再为就绪状态了。但是请注意，如果一直不对这个 fd 做 I/O 操作(从而导致它再次变成未就绪)，内核不会发送更多的通知。该模式与 select 有不同的语义，只有当从不可读变为可读时才触发。上面那种情况，还有 1KB 可读，所以不会触发，当继续读，直到返回 EAGAIN 时，变为不可读，如果再次变为可读就触发。默认可读，调用者可以随便读，直到发生 EAGAIN。可读时读和不读，怎么读都由调用者自己决定，中间 epoll 不管。EAGAIN 后不可读了，等到再次可读，epoll 会再通知一次。理解 ET 模式最重要的就是理解状态的变化，对于监听可读事件时，如果 Socket 是监听 Socket，那么当有新的主动连接到来为状态发生变化；对一般的 Socket 而言，协议栈中相应的缓冲区有新的数据为状态发生变化。但是，如果在一个时间同时接受了 N 个连接($N>1$)，但是监听 Socket 只接受了一个连接，那么其他未 accept 的连接将不会在 ET 模式下给监听 Socket 发出通知，此时状态不发生变化；对于一般的 Socket，如果对应的缓冲区本身已经有了 N 字节的数据，而只取出了小于 N 字节的数据，那么残存的数据不会造成状态发生变化。

(5) epoll 的调用。

epoll 的调用比较简单，只涉及三个函数。

① int epoll_create(int size);

创建一个 epoll 的句柄，size 用来告诉内核这个监听的数目最大值。这个参数不同于 select()中的第一个参数，给出最大监听的 fd+1 的值。需要注意的是，当创建好 epoll 句柄

后，它就是会占用一个 fd 值，所以在使用完 epoll 后，必须调用 close()关闭，否则可能导致 fd 被耗尽。

② int epoll_ctl(int epfd,int op,int fd,struct epoll_event * event);

该函数是 epoll 的事件注册函数，它不同于 select()是在监听事件时告诉内核要监听什么类型的事件，而是在这里先注册要监听的事件类型。

第一个参数是 epoll_create()的返回值。

第二个参数表示动作，用三个宏来表示。

a. EPOLL_CTL_ADD：注册新的 fd 到 epfd 中。

b. EPOLL_CTL_MOD：修改已经注册的 fd 的监听事件。

c. EPOLL_CTL_DEL：从 epfd 中删除一个 fd。

第三个参数是需要监听的 fd。

第四个参数是告诉内核需要监听什么事，数据结构如下：

```
typedef union epoll_data{
void * ptr;
int fd;
__uint32_t u32;
__uint64_t u64
}epoll_data_t;

struct epoll_event {
__uint32_t events;                    /* Epoll events */
epoll_data_t data;                    /* User data variable */
};
```

events 可以是以下几个宏的集合。

a. EPOLLIN ：表示对应的文件描述符可以读(包括对端 Socket 正常关闭)。

b. EPOLLOUT：表示对应的文件描述符可以写。

c. EPOLLPRI：表示对应的文件描述符有紧急的数据可读(这里应该表示有带外数据到来)。

d. EPOLLERR：表示对应的文件描述符发生错误。

e. EPOLLHUP：表示对应的文件描述符被挂起。

f. EPOLLET：将 EPOLL 设为边缘触发模式，这是相对于水平触发来说的。

g. EPOLLONESHOT：只监听一次事件，当监听完这次事件之后，就会把这个 fd 从 epoll 的队列中删除。如果还需要继续监听这个 Socket，需要再次把这个 fd 加入到 epoll 队列中。

③ int epoll_wait(int epfd,struct epoll_event * events,int maxevents,int timeout);

等待事件的产生，类似于 select()调用。参数 events 用来从内核得到事件的集合，maxevents 告诉内核这个 events 有多大，这个 maxevents 的值不能大于创建 epoll_create()时的 size，参数 timeout 是超时时间(毫秒，0 会立即返回，-1 是永久阻塞)。该函数返回需要处理的事件数目，如返回 0 表示已超时。

(6) epoll 的优点。

① 支持一个进程打开大数目的 Socket 描述符。

select 的最大问题是一个进程所打开的描述符是有一定限制的，由 FD_SETSIZE 设置，默认值是 2048。对于那些需要支持的上万连接数目的 IM 服务器来说显然太少了。解决方法：修改这个宏然后重新编译内核，不过这样会带来网络效率的下降；选择多进程的解决方案（传统的 Apache 方案），虽然 Linux 上面创建进程的代价比较小，但依旧不可忽视，加上进程间数据同步远比不上线程间同步的高效，所以也不是一种完美的方案。

epoll 则没有这个限制，它所支持的 FD 上限是最大可以打开文件的数目，这个数字一般远大于 2048。举个例子，在 1GB 内存的机器上是 10 万左右，具体数目可以在 cat /proc/sys/fs/file-max 中查看，一般来说这个数目和系统内存关系很大。

② I/O 效率不随 fd 数目增加而线性下降。

传统的 select/poll 的另一个问题就是当要处理一个很大的 Socket 集合时，由于网络延时，任一时间只有部分的 Socket 是活跃的，但是 select/poll 每次调用都会线性扫描全部的集合，导致效率呈现线性下降。

epoll 不存在这个问题，它只会对活跃的 socket 进行操作——这是因为在内核实现中 epoll 是根据每个 fd 上面的 callback 函数实现的。那么，只有活跃的 Socket 才会主动去调用 callback 函数，其他 idle 状态 Socket 则不会，在这点上，epoll 实现了一个伪 AIO。在一些 benchmark 测试中，如果所有的 Socket 基本上都是活跃的，例如一个高速 LAN 环境，epoll 并不比 select/poll 有什么效率，相反，如果过多使用 epoll_ctl，效率相比还有稍微的下降。但是一旦使用 idle connections 模拟 WAN 环境，epoll 的效率就远在 select/poll 之上。

③ 使用 mmap 加速内核与用户空间的消息传递。

这点实际上涉及 epoll 的具体实现。无论是 select、poll，还是 epoll，都需要内核把 fd 消息通知给用户空间，如何避免不必要的内存复制就很重要，在这点上，epoll 是通过内核与用户空间 mmap 在同一块内存实现的。

④ 内核微调。

这一点其实不算 epoll 的优点，而是整个 Linux 平台的优点。例如，内核 TCP/IP 协议栈使用内存池管理 sk_buff 结构，那么可以在运行时期动态调整这个内存 pool(skb_head_pool)的大小——通过 echo X>/proc/sys/net/core/hot_list_length 完成，其中 X 为要设定的链表长度。再如 listen 函数的第 2 个参数（TCP 完成3 次握手的数据包队列长度），也可以根据平台内存大小动态调整。甚至在数据包数目巨大但同时每个数据包本身很小的特殊系统上，采用 NAPI 网卡驱动架构可进一步提升性能。

17.5.2 Linux OS 级网流处理

Linux OS 级的网流处理主要由 C 语言的 6 个库 libnetcap、libmvutil、libvector 和 jnetcap、jmvutil、jvector，最后编译为动态链接库 libuvmcore. so（详见\src\buildtools\untangle-core. rb 文件）。

libmvutil 将会话事件封装为自定义的 mailbox 结构。

libnetcap 通过 socket 接口从网卡读取网包。

libvector 根据 mailbox 的事件，通知注册到处理流程的观察者（即各安全应用模块），由每个观察者依次对会话进行处理。

libmvutil 引入 epoll 机制。

Untangle VM 对网包的处理流程中需要在 JVM 中调用以上库函数，必须加载：

```
System.loadLibrary("uvmcore");
```

\untangle\src\libnetcap\include\libnetcap. h

定义了主要的 TCP 和 UDP 协议处理函数，以 hook 方式传入。

```
typedef void(* netcap_tcp_hook_t)  (netcap_session_t* tcp_sess, void* arg);
typedef void(* netcap_udp_hook_t)  (netcap_session_t* netcap_sess, void* arg);
```

主要核心文件如下：

\untangle\src\libnetcap\src\netcap_hook. c

\untangle\src\libnetcap\src\netcap_ udp. c

\untangle\src\libnetcap\src\netcap_ tcp. c

\untangle\src\libnetcap\src\netcap_tcp_cli. c

\untangle\src\libnetcap\src\netcap_tcp_srv_complete. c

\untangle\src\libnetcap\src\netcap_sched. c

untangle\src\jnetcap\src\jnetcap. c

17.5.3 Java JVM 级网流处理

对每个 NewSessionRequest 的处理，Untangle 需要同时与 Client 和 Server 建立 sock，每个 session 的建立需要 ClientComplete()和 ServerComplete()同时完成，否则是 half 的连接。这样，通过 UTM 的会话，session 被拆分为源端(src)到 UTM 和 UTM 到目的端(sink)两个会话，拆分过程对于原会话双方是透明的；这对 TCP 不是问题，但是对 UDP 可能会出现问题。

其中 Netcap. java 定义了 Netcap 类，初始化网络流的各项处理。Netcap 类采用 JNI (Java Native Interface)，可以直接调用 libnetcap、libmvutil、libvector 库函数。底层 libnetcap 采用 pthread 多线程。详见 jnetcap. c 文件。

主要核心文件如下：

\untangle\src\jnetcap\impl\com\untangle\jnetcap\Netcap. java

Untangle 系统是基于流 session 处理的。每个 sessionImpl 都有 ArgonAgent，ArgonAgent 是 Node 的一个抽象。

主要核心文件如下：

\untangle\src\jnetcap\impl\com\untangle\jnetcap\NetcapSession. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\SessionGlobalState. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\NetcapTCPSession. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\Session. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\SessionImpl. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\ArgonHook. java

\untangle\\src\uvm-lib\impl\com\untangle\uvm\argon\ArgonAgent. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\ArgonAgentImpl. java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\TCPHook.java

\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\Argon.java

采用 java.util.concurrent 包和 java.nio 包，可以以多线程的方式加快对 Socket 连接的处理。

采用 Key 与 Hash Map 的数据结构及实现方法，可以加快连接维护的速度，便于维持大量的激活并发连接。

17.5.4 Untangle JVM 多线程

核心类

\src\uvm-lib\impl\com\untangle\uvm\engine\UvmContextImpl.java

```
public class UvmContextImpl extends UvmContextBase
    implements LocalUvmContext
{
    ⋮

    public Thread newThread(final Runnable runnable)
    {
        return newThread(runnable, "UTThread-"+ ThreadNumber.nextThreadNum());
}

    ⋮

}
```

每个 Node 都有一个 UvmContextImpl 的类，因此对应有一个线程在运行。下面给出了其中运行的启动线程的代码。

```
uvm-lib\localapi\com\untangle\node\virus\VirusScannerLauncher.java(30):
clam-base\impl\com\untangle\node\clam\ClamScannerClientLauncher.java(79):
openvpn\impl\com\untangle\node\openvpn\OpenVpnCaretaker.java(92):
openvpn\impl\com\untangle\node\openvpn\OpenVpnMonitor.java(252):
openvpn\impl\com\untangle\node\openvpn\VpnNodeImpl.java(213):
webfilter-base\impl\com\untangle\node\webfilter\WebFilterBase.java(230):
router\impl\com\untangle\node\router\DhcpMonitor.java(159):
spam-base\impl\com\untangle\node\spam\RBLChecker.java(166):
spamassassin\impl\com\untangle\node\spamassassin\SpamAssassinScanner.java(47):
uvm-lib\impl\com\untangle\uvm\engine\HeapMonitor.java(152):
uvm-lib\impl\com\untangle\uvm\engine\LogWorker.java(156):
uvm-lib\impl\com\untangle\uvm\engine\NodeManagerImpl.java(611):
uvm-lib\impl\com\untangle\uvm\engine\RemoteToolboxManagerImpl.java(465):
uvm-lib\impl\com\untangle\uvm\engine\RemoteToolboxManagerImpl.java(500):
uvm-lib\impl\com\untangle\uvm\engine\RemoteToolboxManagerImpl.java(533):
uvm-lib\impl\com\untangle\uvm\engine\RemoteToolboxManagerImpl.java(535):
uvm-lib\impl\com\untangle\uvm\engine\RemoteToolboxManagerImpl.java(746):
```

```
uvm-lib\impl\com\untangle\uvm\engine\RemoteToolboxManagerImpl.java(763):
uvm-lib\impl\com\untangle\uvm\engine\UvmContextImpl.java(403):
uvm-lib\impl\com\untangle\uvm\engine\UvmContextImpl.java(405):
uvm-lib\impl\com\untangle\uvm\engine\UvmContextImpl.java(408):
uvm-lib\impl\com\untangle\uvm\engine\UvmContextImpl.java(487):
uvm-lib\impl\com\untangle\uvm\logging\SystemStatus.java(60):
uvm-lib\impl\com\untangle\uvm\logging\SystemStatus.java(61):
uvm-lib\impl\com\untangle\uvm\networking\RuleManager.java(64):
uvm-lib\impl\com\untangle\uvm\networking\RuleManager.java(75):
uvm-lib\localapi\com\untangle\node\util\SimpleExec.java(224):
uvm-lib\localapi\com\untangle\uvm\LocalUvmContext.java(313):
    Thread newThread(Runnable runnable);
uvm-lib\localapi\com\untangle\uvm\LocalUvmContext.java(315):
uvm-lib\localapi\com\untangle\uvm\node\StatisticManager.java(87):
uvm-lib\localapi\com\untangle\uvm\util\JsonClient.java(80):
uvm-lib\localapi\com\untangle\uvm\util\WorkerRunner.java(46):
```

\src\uvm-lib\bootstrap\com\untangle\uvm\engine\Main.java

```
public class Main
{
    private static final String UVM_LOCAL_CONTEXT_CLASSNAME
        ="com.untangle.uvm.engine.UvmContextImpl"

    ⋮
    startUvm()    {
            uvmContext= (UvmContextBase)mcl
                .loadClass(UVM_LOCAL_CONTEXT_CLASSNAME)
                .getMethod("context").invoke(null);

            uvmContext.doInit(this);
    }
    ⋮

}
```

\src\uvm-lib\impl\com\untangle\uvm\engine\Dispatcher.java
\src\uvm-lib\impl\com\untangle\uvm\engine\CronJobImpl.java
\src\uvm-lib\impl\com\untangle\uvm\engine\CronManager.java

17.5.5 Untangle Netcap 多线程

主要核心文件如下：
\untangle\src\uvm-lib\impl\com\untangle\uvm\argon\Argon.java
\untangle\src\jnetcap\impl\com\untangle\jnetcap\Netcap.java
JNI 传递，见 jnetcap.c 中定义，位置如下：

\untangle\src\jnetcap\src\jnetcap. c
\untangle\src\libnetcap\include\libnetcap. h
\untangle\src\libnetcap\src\netcap_thread. c
\untangle\src\libnetcap\src\netcap_server. c

17.6 安全主功能系统

17.6.1 安全功能介绍

所有安全功能组件均集成在 Rack 中，一个 Rack 包括多个 FilterNode 和 ServiceNode。Untnagle Server 同时有多个 Rack，分别对应不同的策略管理。

UVM 上的安全功能组件(Node)分为 FilterNode 和 ServiceNode。

FilterNode 功能组件包括如下。

(1) 网页过滤(Web Filter)。
(2) 防病毒(Virus Blocker)。
(3) 垃圾邮件拦截(Spam Blocker)。
(4) 广告拦截(Ad Blocker)。
(5) 钓鱼网站拦截(Phish Blocker)。
(6) 防间谍软件(Spyware Blocker)。
(7) 防火墙(Firewall)。
(8) QoS 模块。
(9) 入侵保护(Intrusion Prevention)。
(10) 协议控制(Protocol Control)。
(11) HTTPS 协议解剖器。

ServiceNode 服务类模块包括如下。

(1) 虚拟专用网(VPN)。
(2) 攻击拦截(Attack Blocker)。
(3) 报告(Reports)。
(4) 策略管理器(Policy Manager)。
(5) WAN 负载均衡(WANBalancer)。
(6) WAN 故障恢复(WAN Failover)。
(7) 目录管理(Directory Connector)。
(8) 流量记录(Traffic Sniffer)。
(9) 协同防御(Collaborator)。
(10) 网络访问控制(Captive Portal) 。

FilterNode 和 ServiceNode(部分)组件包括每个组件都有启动/关闭按钮，同时有设置按钮(Settings)。

17.6.2 Untangle Shield

Shield 是 Untangle 的 DDoS 保护模块，其基本思路是根据自身的空闲程度接受/拒绝/

调整流量。Shield 的接纳控制是基于 IP 地址的信誉(Reputation)机制。

Shield 的实现分为后台 dameon 进程与前台 UVM 的 Shield Node 管理配置实现。无论是前台还是后台都是以 debian 包安装。前台与后台通过 JSON 接口与 libmicrohttpd 实现的 JSON Server 来交互。

1. Shield daemon 进程

网络启动后所有的 forward 流量进入 NFQUEUE,由 libnetlink 和 libnetfilter_queue 读出包,reader 进行相应的 accept/reject 等操作,同时维护一个 trie 来记录状态以及 LRU 表示最近的活跃的 IP 流等。

Shield 的使用说明如下:

```
~#shield-h                                                    [root@saturn]
Usage: shield
        -d: daemonize.  Immediately fork on startup.
        -p<json-port>: The port to bind to for the JSON interface.
        -c<config-file>: Config file to use.
              The config-file can be modified through the JSON interface.
        -b<bless-file>: User blessing file.
              The bless-file can be modified through the JSON interface.
        -o<log-file>: File to place standard output(more useful with-d).
        -e<log-file>: File to place standard error(more useful with-d).
        -q<queue-num>: Queue to use.
        -l<debug-level>: Debugging level.
        -h: Help(show this message)
```

Shield 启动为守护进程 daemon,由 monitd 进程进行监控,启动的参数显示如下:

```
~#ps aux |grep shield                                     [root@saturn]
root     21688  0.4  0.0   1824   536 ?      S    14:52   0:00 /bin/dash
/usr/share/untangle-shield/bin/monitor.shield
root     21693  0.6  0.0  29420  1392 ?      Sl   14:52   0:00 /usr/bin/shield-p
3001-q 3001-c /etc/untangle-shield.conf-b /var/run/shield-users.conf-o
/var/log/untangle-shield/debug.log-e /var/log/untangle-shield/error.log
```

Shield 启动脚本文件为/etc/init.d/untangle-shield。

主要的配置为/etc/untangle-shield.conf。

Shield 包的源代码见:\untangle\pkgs\untangle-shield\src\main.c。

简单 C 语言实现的 daemon 程序见 http://www.systhread.net/texts/200508cdaemon2.php。

Shield 基于 Netcap 的事件调度实现机制,daemon 的 main.c 函数主要运行了_init(),由_init()函数进行具体的初始化操作。

_init()函数进行一系列初始化操作,例如:

libmvutil_init()

barfight_sched_init()

barfight_bouncer_logs_init()

barfight_shield_init()

barfight_functions_init()

json_server_init()

barfight_net_nfqueue_global_init()

barfight_net_nfqueue_init()

barfight_bouncer_reader_init()

\untnagle\pkgs\untangle-shield\src\functions.c（可以被 json_server 调用的函数）

\pkgs\untangle-shield\src\bouncer\shield.c

\untangle\pkgs\untangle-shield\src\bouncer\reader.c

\untangle\pkgs\untangle-shield\src\bouncer\mode.c

\untangle\pkgs\untangle-shield\src\trie\trie.h

2. Shield UVM node

自动攻击防御 shield 节点包括以下源文件：

\src\shield\api\com\untangle\node\shield\ ShieldNode.java

\src\shield\api\com\untangle\node\shield\ ShieldNodeRule.java

\src\shield\api\com\untangle\node\shield\ ShieldSettings.java

\src\shield\impl\com\untangle\node\shield\ShieldManager.java

ShieldManager 类的 getLogs 将 UVM node 设置中的事件日志，以 json 请求的方式发给 json_server(http://IP：3001)，json_server 再执行 daemon 程序中的 get_logs 函数。

ShieldManager 类中的 blessUsers 将 UVM node 设置中的排除项的规则，以 json 请求的方式发给 json_server(http://IP：3001)，json_server 再执行 daemon 程序中 bless_users 函数。

第 18 章　Untangle Web 界面实现

Web 功能部分是 Untangle 系统的重要部分，它充分体现了 Untangle 系统的易用性。Web 功能部分分为 Apache 部分和 RoR 部分。

18.1　Httpd＋Tomcat 部分

Web 界面系统采用 AJAX 技术，客户端采用 JavaScript 语言。服务器端静态网页主要用 Apache httpd，动态网页基于 Tomcat 架构。动态网页部分用 Java Servlet 和 JSP 技术，后台接 JDBC。

18.1.1　Apache Httpd 服务器

Apache 的配置参照/etc/apache2/sites-enabled/uvm@符号链接，即以下文件目录：

/etc/apache2/sites-available/uvm

/etc/apache2/httpd.conf

/etc/apache2/untangle-conf.d/uvm.conf

/etc/apache2/untangle-conf.d/workers.properties

/etc/apache2/mods-available/jk.conf

/usr/share/untangle/web/webui/index.jsp

```
RedirectMatch 301 ^/webstart.*  /webui/startPage.do

<VirtualHost *:80>
    Include /etc/apache2/untangle-conf.d/*
    Include /etc/apache2/untangle-unrestricted-conf.d/*
</VirtualHost>

<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/apache.pem
    Include /etc/apache2/untangle-conf.d/*
    Include /etc/apache2/untangle-unrestricted-conf.d/*
</VirtualHost>

<VirtualHost *:64156>
    Include /etc/apache2/untangle-unrestricted-conf.d/*
</VirtualHost>

<VirtualHost *:64157>
```

```
    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/apache.pem
    Include /etc/apache2/untangle-conf.d/*
    Include /etc/apache2/untangle-unrestricted-conf.d/*
</VirtualHost>
```

UntangleWeb 网页位置如下：

```
/var/www/
/var/www/UntangleTk
/var/www/AjaxTk
/var/www/ext
/var/www/java/
/var/www/docs/
/var/www/jsonrpc/
/var/www/scripts/
```

18.1.2 Apache Tomcat 服务器

1. Apache Tomcat 基础

Tomcat 是一款功能强大的 Servlet 容器，为 Servlet 提供了执行环境以及系统资源（如文件系统）。Tomcat 由插件组成，插件以嵌套的方式装配在一起。组件的配置说明服务如何运行，包括是否使用特定的过滤器、服务器监听的端口等。

访问 Servlet 的推荐途径主要是逻辑映射（Logical Mapping），映射 URL 的文件路径到 Servlet 上。另一种是对 Servlet 匹配其字符序列。

虽然 Servlet 比 CGI 完善，但其主要用途为逻辑处理，在创建 HTML 文件方面不太实用，要求 HTML 设计人员要熟悉 Java，以避免破坏 Servlet。为此引入 JSP 技术。

JSP（Java Server Page）是在传统的网页中加入 Java 程序段与 JSP 标签。Java 程序段用于操作数据库，重定向网页和发送邮件。Tomcat 服务器接收到 Web 客户端的 JSP 请求时，对 JSP 语法分析生成 Java Servlet。Mod_jk 负责与 Tomcat 通信，Tomcat 作为 Servlet 的容器生成动态网页内容。Tomcat 的 worker 程序等待执行 Sevlet 的 Tomcat 实例，从 Apache Web Server 转发对应 Servlet 请求到 worker 程序。

Tomcat 包含 Catalina 的 Servlet 容器用于执行 Servlet 和编译 JSP，还包含 JSP 文件的编译器 Jasper。JSP 编译器与 Servlet 容器的整合统称为 Web 容器（即可以整合 Java Web 应用的容器）。

每个标签都有一个相应的 Java 类，包含了可能出现在网页上的其他代码。在网页中标签看上就像 HTML 一样，有一个开始标签，接着是一个结束标签，以及指定的文本。这些标签的生命周期包含一个方法，它在开始标签激活时调用，叫做 doStartTag()；另一个方法在结束标签时被激活调用，叫做 doEndTag()；还有一个标签在下一个请求就绪时转换状态时调用。

JSP EL (JSP 扩展语言)定义了一种易于使用的语法用于访问 Java Beans 等。

2. Apache Tomcat 启动

src\uvm-lib\impl\com\untangle\uvm\engine\TomcatManager.java

其核心类的 TomcatManager 的参数如下：

```
TomcatManager(UvmContextImpl uvmContext,
                InheritableThreadLocal<HttpServletRequest> threadRequest,
                String catalinaHome, String webAppRoot, String logDir)
```

src\uvm-lib\impl\com\untangle\uvm\engine\UvmContextImpl.java

启动 Tomcat 服务，设置 Tomcat 参数：

```
tomcatManager=new TomcatManager(this, threadRequest,
                                System.getProperty("bunnicula.home"),
                                System.getProperty("bunnicula.web.dir"),
                                System.getProperty("bunnicula.log.dir"));
```

即把 bunnicula. home 设置为/usr/share/untangle，bunnicula. web 设置为/usr/share/untangle/web。

```
// create an Catalina Engine
                Engine baseEngine=emb.createEngine();

                // set Engine properties
                baseEngine.setName("tomcat");
                baseEngine.setDefaultHost(hostname);

                loadSystemApp("/blockpage", "blockpage");
                loadSystemApp("/reports", "reports",new WebAppOptions(true,
                                                new ReportingOutsideAccessValve()));
                loadSystemApp("/alpaca", "alpaca",…);
                loadSystemApp("/webui", "webui",…);
                loadSystemApp("/setup", "setup",…);
                loadSystemApp("/library", "library",…);
```

src\uvm-lib\bootstrap\com\untangle\uvm\engine\Main.java

src\uvm-lib\impl\com\untangle\uvm\engine\AppServerManagerImpl.java

Tomcat 目录包括主目录 CATALINA_HOME 和/bin、/conf、/lib、/logs、/tmp、/web、/work，如下所示。

```
$CATALINA_HOME   Tomcat 安装目录
├─bin           用以启动、关闭 Tomcat 或者其他功能的脚本(.bat 文件和 .sh 文件)
├─common        Catalina 和 Web 应用程序用到的 Class 及库文件
├─conf          用以配置 Tomcat 的 XML 及 DTD 文件
├─logs          Catalina 和其他 Web 应用程序的日志
├─server        Catalina 用到的 Class 及库文件
├─shared        Web 应用程序用到的 Class 及库文件
├─temp          临时文件目录
├─webapps       Web 应用程序根目录
└─work          用以生成由 JSP 编译生成的 Servlet 的 .java 和 .class 文件
```

work 目录用于工作中和临时文件，当 JSP 编译过程中 JSP 转换为 Javaservlet 并通过该目录被访问，该目录将会大量被使用。

web 目录是 Web 应用程序的存放位置，标准的 Tomcat 带有几个应用程序在该目录中。

web 目录包括如下目录：

```
/alpaca/  blockpage/  idblocker/  library/  openvpn/  quarantine/  reports/  setup/  spyware/
virus/  webfilter/  webstart/  webui/
```

其中，webui 对应着外部访问为

```
http://IPaddr/webui
/usr/share/untangle/web/webui/WEB-INF/web.xml
```

Untangle 的 Servlets 包括：

```
src/uvm-lib/servlets/alpaca
src/uvm-lib/servlets/blockpage
src/uvm-lib/servlets/library
src/uvm-lib/servlets/reports
src/uvm-lib/servlets/setup
src/uvm-lib/servlets/webui
```

不同的安全组件的设置采用 Hibernate 作为一个对象持久化到数据库中(Persistence)。详见安全应用 Node 开发过程。

3. Untangle-VM Web 部署

Web 应用程序一般安装在<$CATALINA_HOME/webapps>目录下，Serverlet 2.5 要求 Web 应用遵从一定的基础目录架构。

Web 应用程序位于/webapps 中，通过 URL：http://localhost/webapps/可以访问到。/webapps/也称为 Web 应用程序的 context 路径。

Web 应用程序架构需要含有 WEB-INF 目录的 web.xml 文件。WEB-INF 和 META-INF 目录是应用程序的私有资源，不能被直接被客户端应用程序直接访问。

META-INF 目录可以包含两个配置文件：manifest 文件(manifest.xml)和 context 文件(context.xml)。

WEB-INF 和 META-INF 目录之外的一切资源都是公共资源，可以通过相应的 URL 访问。

目录结构如下：

```
$CATALINA_HOME/conf/ web.xml
$CATALINA_HOME/web/
                         |------------webui/index.jsp,blank.html, build.xml, hello.html
                                      |------------images/
                                      |------------script/ JavaScript files
                                      |------------META-INF
                                      |------------WEB-INF/web.xml
```

```
|-------------classes/
|-------------lib/
|-------------tags/
```

一般情况，当从浏览器请求 Web 资源时，Web 服务器将提供静态的 Web 资源。但是浏览器在请求 JSP 页面时，需要将 JSP 页面编译为 Java 文件，再编译 Java 文件为 Servlet 类，生成 class 文件，最后运行在浏览器上输出结果。

任何以 jsp 结尾的 URL 应当被传送到名为 JSP 的 Servlet，该 Servlet 在 $CATALINA_HOME/conf/web. xml 配置文件中定义。可以看到 Servlet 的完全名称为 org. apache. jasper. servlet. JspServlet。请求传递的 Servlet 使用 context 路径来加载 JSP 页面并将其传递到 Tomcat 的 JSP 编译器，如 Jasper。load-on-startup 选项保证在启动过程中 Servlet 类以优先级 1 被装入内存，以保证它在任何 JSP 请求前被加载。

Default Servlet 用来为所有的 Web 应用的静态资源内容(HTML/GIF 文件)服务，也为目录服务。换言之，Default Servlet 提供相当于标准的 Web 服务的能力。

Invoker Servlet 能直接加载或者执行任何的 Servlet，因为它能够像任何一个 Servlet 发出请求。Invoker Servlet 有可能成为系统中的安全隐患。

JspServlet 将 JSP 网页归到 Servlet 中执行。SSI 和 CGI Servlets 配置就不再叙述了。

```
<servlet>
  <servlet-name>default</servlet-name>
  <servlet-class>
    org.apache.catalina.servlets.DefaultServlet
  </servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <init-param>
    <param-name>listings</param-name>
    <param-value>false</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<!--
    <servlet>
    <servlet-name>invoker</servlet-name>
    <servlet-class>
    org.apache.catalina.servlets.InvokerServlet
    </servlet-class>
    <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
    </init-param>
```

```
    <load-on-startup>2</load-on-startup>
    </servlet>
-->

<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
  <init-param>
    <param-name>fork</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>xpoweredBy</param-name>
    <param-value>false</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>
```

通过使用<servlet-mapping>元素定义的 URL mapping 来实现这些操作。在 $CATALINA_HOME/conf/web.xml 中定义了该 URL mapping。该文件是 Web 应用程序的部署描述符，独立的 Web 应用可以自己定义自己的描述符。

```
<!--================Built In Servlet Mappings=========================-->


<!--The servlet mappings for the built in servlets defined above.  Note        -->
<!--that, by default, the CGI and SSI servlets are not mapped.  You           -->
<!--must uncomment these mappings(or add them to your application's own       -->
<!--web.xml deployment descriptor)to enable these services                    -->

<!--The mapping for the default servlet-->
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<!--The mapping for the invoker servlet-->
<!--
  <servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
-->

<!--The mapping for the JSP servlet-->
<servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>*.jsp</url-pattern>
```

```
</servlet-mapping>

<servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>*.jspx</url-pattern>
</servlet-mapping>

<!--The mapping for the SSI servlet-->
<!--
    <servlet-mapping>
    <servlet-name>ssi</servlet-name>
    <url-pattern>*.shtml</url-pattern>
    </servlet-mapping>
-->

<!--The mapping for the CGI Gateway servlet-->

<!--
    <servlet-mapping>
    <servlet-name>cgi</servlet-name>
    <url-pattern>/cgi-bin/*</url-pattern>
    </servlet-mapping>
-->
```

4. JavaScript 客户端

浏览器的 Web 界面系统采用 AJAX 技术，$CATALINA_HOME/webapps/script 中存放基于 ExtJS 开发的 JavaScript 的界面功能。

JavaScript 文件主要有：

```
main.jsmain-min.js components.jscomponents-min.jsmd5.jsmd5-min.js
```

untangle-node-*/

```
*.settings*-min.js
```

config/

```
administration.js      email.js      localDirectory.js      policyManager.js
systemInfo.js      system.js      upgrade.js
administration-min.js      email-min.js      localDirectory-min.js      policyManager-min.js
systemInfo-min.js  system-min.js  upgrade-min.js
```

*-min.js 为去掉冗余字符的压缩版本。

18.2 RoR 部分

Untangle 系统的 untangle-net-alpaca 和 untangle-restore-tools 是基于 Ruby on Rails (ROR)架构，RoR 的源文件为 pkgs/untangle-rails。

单击 Config，再单击 networking，出现配置界面 http://localhost/alpaca，在网页上进行配置。

```
http://localhost:3000/alpaca
```

用户名为 root，密码为管理员账号。

```
http://localhost:3000/alpaca/interface/list
```

单击 Config，再单击 networking，出现配置界面 http://localhost/apaca，在网页上的 Basic Mode 的 wizard 进行配置。

```
http://localhost:3000/alpaca/wizard
```

1. JavaScript 客户端

以流量控制(Traffic Control)为例：

```
pkgs/untangle-net-alpaca/files/var/lib/rails/untangle-net-alpaca/public/
javascripts/e/rule_builder.js
```

rule_builder.js 描述了客户端 qos 规则创建的面板。

```
pkgs/untangle-net-alpaca/files/var/lib/rails/untangle-net-alpaca/public/
javascripts/e/glue.js
```

saveSettings，参数为 panel confirmedSaveSettings，保存设置，发送 HTTP 请求到 ruby，请求链接为 saveSettings 或 saveMethod。

```
pkgs/untangle-net-alpaca/files/var/lib/rails/untangle-net-alpaca/public/
javascripts/e/util.js
```

util.js 中 executeRemoteFunction 方法调用 ruby，参数通过 json 格式传递，ruby 中可以直接以对象的形式调出，如 params['bandwidth']，params['rules']…

2. Rails 服务器端

Mongrel 是 Rails 的默认 Web 服务器。http://localhost：3000 用于判断 Mongrel 服务器是否已经启动。

untangle-net-alpaca 负责网络配置，包括界面以及相应的脚本。untangle-net-alpaca 采用 Rails 架构，基于 Mongrel 服务器。其 Web 目录为

```
/var/www/UntangleTk
/var/www/AjaxTk
/var/www/ext
/var/www/java/
Var/www/docs/
Var/www/jsonrpc/
var/www/scripts/
```

启动/etc/init.d/untangle-net-alpaca 脚本，也就启动了 Web 服务器 mongrel_rails.

启动/etc/init.d/untangle-net-alpaca-iptables 脚本，将运行/etc/ untangle-net-alpaca/

scripts/iptables 脚本，该脚本运行了 nf 的 load_modules(/usr/share/untangle-net-alpca/scripts/load_modules)。

默认的加载 netfilter 模块有：

```
DEFAULT_MODULES="nf_conntrack \
        nf_conntrack_h323           nf_conntrack_netbios_ns \
        nf_conntrack_proto_gre      nf_conntrack_tftp\
        nf_conntrack_amanda         nf_conntrack_ipv4\
        nf_conntrack_netlink        nf_conntrack_proto_sctp\
        nf_conntrack_ftp            nf_conntrack_irc\
        nf_conntrack_pptp           nf_conntrack_sip     nf_nat               nf_nat_ftp\
        nf_nat_irc                  nf_nat_proto_gre     nf_nat_snmp_basic    nf_nat_amanda\
        nf_nat_h323                 nf_nat_pptp     nf_nat_sip           nf_nat_tftp"
```

然后该脚本的 run_iptables_scripts()函数逐个运行/etc/ untangle-net-alpaca/iptables-rules.d/下的 iptables 规则。

最后该脚本运行(将转发的网包送入 NFQUEUE 10 号队列)：

```
./sbin/iptables -A FORWARD -j NFQUEUE --queue-num 10
```

注意：

(1) 在 Linux 2.4 内核中出现了 ip_queue，用于将网包从内核空间传递到用户空间，缺点是只能有一个应用程序接收内核数据。Linux 2.6.14 以后，新增了 nfnetlink_queue，理论上可最大可支持 65 536 个应用程序接口，而且可以兼容 ip_queue。但是从内核到用户空间的通道还是只有一个，实际上 netfilter 对每个协议族也只有一个队列，这里说的 65 536 个子队列的实现就像 802.1Q 实现 VLAN 一样是在网包中设置 ID 号来区分的，不同 ID 的包都通过相同的接口传输，只是在两端根据 ID 号进行了分类处理。

(2) 用户空间。用户空间的支持库包括两个：libnfnetlink 和 libnetfilter_queue，后者需要前者支持，其源码见 netfilter 项目网站。

(3) 用户空间与 QUEUE 的关系。Untangle 需要驻留在 kernel 里的 netfilter 把途经本机的网包放进 queue 里，由在 user mode 里的 UVM 使用 libnetfilter_queue 库收取这些网包。

联网的基本模式的配置向导，其代码采用 rails 框架，源代码在如下目录中：

(http://localhost:3000/alpaca/wizard)的 Rails 框架架构在

```
/var/lib/rails/untangle-net-alpaca/lib/
/var/lib/rails/untangle-net-alpaca/lib/alpaca/
```

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/network_manager.rb

```
更新 /etc/network/interfaces
更新 /etc/network/run/ifstate
更新 /etc/untangle-net-alpaca/ethernet_media
```

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/qos_manager.rb

更新 /etc/untangle-net-alpaca/tc-rules.d/100-high-priority*
更新 /etc/untangle-net-alpaca/tc-rules.d/200-mid-priority*
更新 /etc/untangle-net-alpaca/tc-rules.d/300-low-priority*
更新 /etc/untangle-net-alpaca/tc-rules.d/900-system-priority

更新 /etc/ untangle-net-alpaca/iptables-rules.d/800-qos

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/packet_filter_maneger. rb

更新 /etc/ untangle-net-alpaca/iptables-rules.d/010-flush
更新 /etc/ untangle-net-alpaca/iptables-rules.d/100-init-chains
更新 /etc/ untangle-net-alpaca/iptables-rules.d/200-networking
更新 /etc/ untangle-net-alpaca/iptables-rules.d/400-firewall
更新 /etc/ untangle-net-alpaca/iptables-rules.d/600-redirect
更新 /etc/ untangle-net-alpaca/iptables-rules.d/700-nat-firewall
更新 /etc/ untangle-net-alpaca/iptables-rules.d/900-single-nic

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/uvm_manager. rb

更新 /etc/untangle-net-alpaca/iptables-rules.d/ 800-uvm
更新 /etc/untangle-net-alpaca/iptables-rules.d/ 475-uvm-services(before the firewall)
更新 /etc/untangle-net-alpaca/iptables-rules.d/ 675-uvm-services(after the firewall)
更新 /etc/untangle-net-alpaca/iptables-rules.d/ 475-openvpn-pf
更新 /etc/untangle-net-alpaca/interface.properties
更新 /usr/share/untangle-net-alpaca/scripts/uvm/update-configuration

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/routes_maneger. rb

更新 /etc/ untangle-net-alpaca/routes(路由信息可以由以下命令显示)：
#ip route show table all type unicast

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/routes_maneger. rb

更新 /etc/ untangle-net-alpaca/scripts/update-address.d/10-hostname

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/hostname_manager. rb

UpdateHostNameScript="/etc/untangle-net-alpaca/scripts/update-address.d/10- hostname"

/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/dns_server_manager. rb

UpdateHostNameScript="/etc/untangle-net-alpaca/scripts/update-address.d/11- dnsmasq-hosts"

pkgs\untangle-net-alpaca\files\var\lib\rails\untangle-net-alpaca\app\controllers\qos_controller.rb

qos_controller. rb 定义了 ruby 的 HTTP Server 的后台，如 get_settings 和 set_settings，set_settings 调用各配置（Active_Record 子类）的 save 方法保存配置，并在最后调用 os['packet_filter_manager']的 commit 方法，更新 iptables 规则，该类定义在 packet_filter_manager. rb。

pkgs\untangle-net-alpaca\files\var\lib\rails\untangle-net-

```
alpaca\lib\os_library\debian\packet_filter_manager.rb
pkgs\untangle-net-alpaca\files\var\lib\rails\untangle-net-alpaca\lib\os_library\debian\*_
manager.rb
```

基本上是在启动或者更新设置时，把添加规则(所有规则，包括新规则和原有规则)的命令写到脚本文件/etc/untangle-net-alpaca/xxx 中(如/etc/untangle-net-alpaca/untangle-qos/*，/etc/untangle-net-alpaca/tc-rules. d/*，前两个在 qos_manager. rb 中，/etc/untangle-net-alpaca/iptables-rules. d/*，后一个在 packet_filter_manager. rb 中)，再运行应用，使新规则集生效(TC 脚本采用 Linux LARTC 的代码，见 http://lartc. org/wondershaper)。

3. 网络配置初始化脚本

http://localhost:3000/net-alpaca

```
/etc/init.d/networking
/etc/network/if-up.d/alpaca(run-parts /etc/ untangle-net-alpaca/scripts/update-address.d/
/etc/network/if-down.d
/etc/network/if-post-down.d/
/etc/network/ if-pre-up.d/
/etc/network/run/
 /etc/network/interfaces
```

所有网络设备均在 /sys/class/net/。

DHCP Server 的配置文件在/var/lib/dhcp/dhcpd. lease。

```
/var/lib/rails/untangle-net-alpaca/app/controllers/dhcp_controller.rb
/var/lib/rails/untangle-net-alpaca/lib/os_library/null/dhcp_manager.rb
/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/dhcp_manager.rb
/var/lib/rails/untangle-net-alpaca/lib/os_library/debian/dhcp_server_manager.rb
/var/lib/rails/untangle-net-alpaca/lib/os_library/dhcp_manager.rb
/var/lib/rails/untangle-net-alpaca/lib/os_library/dhcp_server_manager.rb
```

第 19 章　Untangle uvm 数据库系统

Untangle 基于开源 PostgreSQL 关系型数据库,并把该数据库作为其系统设置、系统事件和用户资料的存储。PostgreSQL 关系型数据库以性能和稳定性著称。

19.1　PostgreSQL 数据库

启动 postgresql:

```
#service postgresql start
#/etc/init.d/postgresql start
```

查看 postgresql

```
#psql uvm postgres(psql dbname username)
```

进入 postgresql 命令行交互

```
uvm=#
```

提示符=#表明目前是数据库超级用户,最可能出现在安装 PostgreSQL 的用户情况下,作为超级用户意味着不受访问控制的限制。

19.2　PostgreSQL 基本命令

基本命令如下。

\q:退出。

\l:查看系统中现有数据库。

\c:从一个数据库转到另一个数据库。

\dt:查看表。

\d:查看结构。

\di:查看索引。

基于\dt 命令显示的 Untangle PostgreSQL 数据库 uvm 列表详见附录 B,部分列表详见以下查询结果:

```
uvm=#select version();
                                    version
-------------------------------------------------------------
PostgreSQL 8.3.7 on i486-pc-linux-gnu, compiled by GCC gcc-4.3.real(Debian 4.3.2-1.1)4.3.2
(1 笔资料列)
```

```
uvm=#select current_date;
uvm=#select * from uvm_branding_settings;
uvm=#select * from n_mail_settings;
uvm=#select * from u_policy;
uvm=#select * from pl_stats;
(9 笔资料列)

uvm=#select * from pl_endp;
uvm=#select * from u_address_settings;
 settings_id | https_port |       hostname        | is_hostname_public | has_public_address |
public_ip_addr | public_port
-----------+---------+-------------+-------------+---------------
+---------------+-------------
      2496619|          443| untangle.example.com | f                              | f
|                        |             443
(1 笔资料列)

uvm=#select * from n_shield_settings;
settings_id | tid
-----------+------
          8962 |    16
(1 笔资料列)

uvm=#select * from u_node_manager_state;
id | last_tid
--+-----------
  1 |          20
(1 笔资料列)

uvm=#select * from n_ips_settings;
settings_id | max_chunks | tid
-----------+---------+------
           5261|               8 |     9
(1 笔资料列)

uvm=#select * from u_login_evt;
uvm=#select * from n_sniffer_evt;
   event_id |          time_stamp             |
message
---------+------------------+-------------------------------
------------------------------------------------
     95510 | 2010-08-13 19:58:08.146 | query: query to_file "32855786.pcap" index ip
"0.0.0.0"
   2236142 | 2010-08-23 19:52:35.073 | query: query to_file "76823071.pcap" index ip
"192.168.1.126" start 1281974400 end 1282564321
```

```
2236143 | 2010- 08- 23 19:53:24.194 | query: query to_file "47362689.pcap" index ip
"192.168.1.126" start 1281542400 end 1282564321
2236144 | 2010- 08- 23 19:53:59.803 | query: query to_file "78734174.pcap" index connection2 "tcp 192.168.
1.126 0.0.0.0" start 1281542400 end 1282564321
2236147 | 2010- 08- 23 19:56:57.253 | query: query to_file "2362052.pcap" index ip
"192.168.1.2"
2236148 | 2010- 08- 23 19:57:24.068 | query: query to_file "88824083.pcap" index ip
"10.0.6.47"
(6 笔资料列)
```

19.3 Untangle 数据库 uvm

数据库初始化源代码见：\untangle\src\uvm-lib\bootstrap\com\untangle\uvm\engine\DataSourceFactory.java。

```
public class DataSourceFactory
{
    private static final DataSourceFactory FACTORY= new DataSourceFactory();

    private final ComboPooledDataSource dataSource;

    private DataSourceFactory()
    {
        dataSource= new ComboPooledDataSource();
        try {
            dataSource.setDriverClass("org.postgresql.Driver");
        } catch(PropertyVetoException exn){
            throw new RuntimeException(exn); // won't happen
        }
        dataSource.setJdbcUrl("jdbc:postgresql://localhost/uvm?charset= unicode");
        dataSource.setUser("postgres");
        dataSource.setPassword("foo");
        dataSource.setMaxStatements(180);
        dataSource.setMinPoolSize(5);
        dataSource.setMaxPoolSize(50);
        dataSource.setMaxIdleTime(300);
        dataSource.setTestConnectionOnCheckout(true);
        dataSource.setPreferredTestQuery("SELECT 1");
    }

    // public factories-----------------------------------------------

    public static DataSourceFactory factory()
    {
        return FACTORY;
    }
```

```
        ⋮
}
```

数据库中主要保存 settings 和 events 两大类。

settings 类的数据表主要由各 node 的 * Settings. java 进行持久化操作实现，比如 AdBlockerSettings. java。

\untangle\src\adblocker\api\com\untangle\node\adblocker\AdBlockerSettings. java

```
/**
 * Hibernate object to store Ad Blocker settings.
 *
 * @version 1.0
 */
@Entity
@Table(name="n_adblocker_settings", schema="settings")
public class AdBlockerSettings implements Serializable {

    private Long id;
    private Tid tid;

    private AdBlockerBaseSettings baseSettings=new AdBlockerBaseSettings();
    private Set<StringRule> rules=new HashSet<StringRule>();

    private Set<IPMaddrRule> passedClients=new HashSet<IPMaddrRule>();
    private Set<StringRule> passedUrls=new HashSet<StringRule>();

    public AdBlockerSettings(){
    }

    public AdBlockerSettings(Tid tid){
        this.tid=tid;
    }

    @Id
    @Column(name="settings_id")
    @GeneratedValue
    public Long getId(){
        return id;
}

        ⋮
}
```

events 类的数据表主要由各 node 的 * Event. java 进行持久化操作实现，比如 AdBlockerEvent. java。

第 20 章　Untangle 运行管理

20.1　Untangle 运行监控

monit 是 UNIX 管理监控进程、文件、目录和文件系统的工具。monit 可以进行自动维护、修复和执行错误发生时的有意义的后续动作，如重启失去响应的进程或者占用过多资源的进程，如文件目录和文件系统的时戳改变、校验改变和大小改变。monit 还提供 HTTP 和浏览器接口（https：//localhost：2812/）。

monit 由配置文件进行管理，一般放在 monit.d 目录下，如/etc/monit.d/conf。

```
pkgs\untangle-monit-config\files\etc\untangle\monit.conf
```

对各个组件的监控可由相应的 *.conf 文件来进行管理，配置文件如下：

```
monit
pkgs\untangle-monit-config\files\etc\untangle\monit.d\monit-base_all.conf
Apache
pkgs\untangle-apache2-config\files\etc\untangle\monit.d\untangle-apache2.conf.template
ClamAV
pkgs\untangle-clamav-config\files\etc\untangle\monit.d\clamav_i3820.conf
Spamassassin
pkgs\untangle-spamassassin-update\files\etc\untangle\monit.d\spamassassin_all.conf
Slapd
pkgs\untangle-ldap-server\files\etc\untangle\monit.d\slapd_all.conf
Postgresql
pkgs\untangle-postgresql-config\files\etc\untangle\monit.d\postgresql_all.conf
```

20.2　Untangle 管理工具集

20.2.1　Untangle 本地管理

Untangle 本地管理界面和工具主要是在/usr/share/untangle-kiosk/homes/kiosk/utils/。

```
ut-desktop.sh
ut-off.sh   关机
ut-reboot.sh   重启
screensaver.sh
ut-resolution.sh
ut-shell.sh   封装 Linux shell
```

ut-restore.sh　恢复界面

平台恢复界面主要有以下选项：

Untangle platform Recovery
Backup&Restore
Return to Factory Defaults
Reset Administrative Accounts
Upgrade Untangle platform
Remote Support

注意：如果忘记 Web 管理界面，可以通过 shell 登录为 root 用户，可以用 ut-restore.sh 将 Web 登录 portal 恢复成默认设置 admin/passwd。

本机管理界面和工具位置目录如下。

/usr/share/untangle/bin

backup-ut*	get-last-update*	net-properties*	update-schema*
banner-nanny.sh*	inspect_ca*	phish-get-last-update*	utactivate*
bunnicula*	interface-properties*	purgedb*	ut-backup-bundled.sh*
clam-get-last-update*	kill_all_sessions*	rup*	uterr*
createpopid.rb*	list-packages.sh*	spamassassin-get-last-update*	utip*
dhcp-lease-list*	log-mem*	untangle-clamav-config*	utregister*
dns-static-host-list*	migrate-postgres.sh*	untangle-reports-wrapper.sh*	ut-remotebackup.sh*
factory-defaults*	mkg*	untangle-restore*	ut-restore*
fixkeyboard*	monit-wrapper.sh*	untangle-snmp.py	ut-restore-bundled.sh*
uttimezone*	utuptime*	uvmdb-restore*	version_newest.sh*
xpd*			

20.2.2　Untangle 外部管理配置

需要登录本机系统 Web 管理桌面(默认命令是 admin/passwd)。在管理界面中，进入配置→管理，添加管理员账号。在外部管理中启用外部管理，进行外部报告查看，允许来自任何 IP 地址的外部访问。

20.2.3　Untangle 外部 SSH 登录管理

Untangle 初始化 root 用户密码需要在本机系统桌面单击终端，按提示输入 root 用户密码。

为了适合远程管理，可启用 SSH 服务。启用 SSH 服务：删除/etc/ssh/sshd_not_to_be_run 文件，并执行/etc/init.d/ssh start 启用 SSH 服务，若需关闭 SSH 服务，只需重新建立空文件/etc/ssh/sshd_not_to_be_run 即可。

20.2.4　Untangle 命令行工具

UCLI(Untangle Command Line Input) 是非常有用的管理工具。

/usr/share/untangle/pycli/jscli.py Usage:

```
optional args:
  -h hostname
  -u username
  -w password
  -t timeout (default 120000)
  -p policy
  -v
toolbox commands:
  ucli install mackage-name
  ucli uninstall mackage-name
  ucli update
  ucli upgrade
  ucli requestInstall mackage-name
toolbox lists:
  ucli available
  ucli installed
  ucli uninstalled
  ucli upgradable
  ucli uptodate
node manager commands:
  ucli instantiate mackage-name [ args ]
  ucli start TID
  ucli stop TID
  ucli destroy TID
  ucli neverStarted
node manager lists:
  ucli instances
node live sessions:
  ucli sessions [ TID ]
admin manager:
  ucli who
  ucli getRegInfo
  ucli passwd [-a |-d ] login [ password ]
uvm commands:
  ucli shutdown
  ucli serverStats
  ucli gc
  ucli loadRup
  ucli setProperty key value
policy manager:
  ucli addPolicy name [notes]
  ucli listPolicies
reporting manager:
  ucli isReportingEnabled
  ucli areReportsAvailable
```

```
    ucli prepareReports [ args ]
    ucli startReports
    ucli stopReports
  logging manager:
    ucli userLogs tid
    ucli resetLogs
    ucli logError [text]
  apt commands:
    ucli register mackage- name
    ucli unregister mackage- name
  argon commands:
  nucli server commands:
    ucli restartCliServer
  debugging commands:
    ucli aptTail
```

运行 jucli. py，通过 pyCurl，向认证 URL 传输认证信息(parser. username、parser. password)，先进行认证。

认证 URL 为

http://localhost/webui/JSON-RPC。

http://hostname/auth/login。

20.3 Untangle 日志

Untangle 的日志目录为

```
/var/log/untangle- net- alpaca
/var/log/untangle- shield
/var/log/uvm
```

20.3.1 Java Virtual Machine 监控

1. CPU 利用率

```
#top (shift+ H)+ ThreadDump
按 1键后,显示多核处理器上的每个核的利用率.

#pidstat  -p [PID]- t 1 5
```

2. 文件 I/O 消耗

```
#pidstat  -d- t- p [PID] 1 100
#iostat- x xvda 3 5
```

3. 网络 I/O 消耗

```
#sar-n ALL 1 2
```

4. 内存消耗

```
#sar-r
#pidstat-r-p [pid] [interval] [times]
```

5. JVM 内存状况

```
#jmap-heap 5084(显示 JVM中各级内存分布状况)
#jmap-histo 5084(显示 JVM中对象内存占用状况)
#jmap-dump:format=b,file=filename 5084(导出整个 JVM的内存信息)
比如 #jmap-dump:format=b,file=Saturn.bin 5084
Eclipse Memory Analyzer用来显示 dump的文件。
```

6. 编译与类加载

```
#jstat-class pid
#jstat-printcompilation pid
#jstat-gcutil pid
```

7. JVM 线程运行状况

```
#jstack pid
#pstree-a
```

8. JVM 配置信息

```
#jinfo pid(JVM配置信息)

#jps-l-v-V(查看 JVM进程状态)
```

9. jnettop

```
#jnettop
```

20.3.2 Untangle 统计日志

系统统计日志记录可统计系统信息,如磁盘使用情况,代码位于\src\uvm-lib\impl\com\untangle\uvm\logging\SystemStatus.java 中。

```
public class SystemStatus {
    private String _buildDynamicStat()    {
        ⋮
        proc=LocalUvmContextFactory.context().exec("/bin/df-h");
        ⋮
}
    private String _buildUVMStat(){
    ⋮
}
```

20.4 Untangle 系统自检与测试

系统可自检自身功能的正确性。

/usr/share/untangle/tests/

untangle-base-virus/ untangle-node-phish/ untangle-node-spyware/ untangle-monit-config/ untangle-postgresql-config/

untangle-base-webfilter/ untangle-node-reporting/ untangle-casing-mail/ untangle-net-alpaca/

untangle-vm/

20.5 硬件看门狗

为了保障高可靠性，除了旁路直通功能外，还需要硬件看门狗功能（Watchdog）支持。初始设计硬件看门狗功能的基本用途是控制系统重启，可以在主板硬件中设定，也可以通过监护程序控制。监护程序周期性不间断地执行“喂狗”动作，如在一个周期内，不断给定时器刷新时间，看门狗就不发出重启指令。如果软件系统故障时，如系统宕机时，监护程序不能执行“喂狗”动作，硬件看门狗则发出系统重启指令。

Watchdog 模块与 Linux 内核一同编译，并在系统启动时加载。请参考 Linux 内核驱动程序设计。

第 21 章　Untangle 开发测试调试

21.1　Untangle 源代码

Untangle 代码结构包括三部分。

(1) Debian Linux 内核修改版。

(2) Untangle Virtual Machine 代码。

(3) 附属部分：基于 Xwindows，一个简单本机图形化界面。

21.1.1　总体结构

Untangle 的代码主要分为三个目录：untangle/kernels、untangle/pkgs 和 untangle/src。

1. Untangle/kernels

kernels 目录下存放的是 Linux 内核(2.6.26)源码，以及 Untangle 需要的 netfilter 补丁包。对内核修改后，最终可以编译 deb 包进行安装重启生效。

2. Untangle/pkgs

pkgs 目录中，比较关键的与网络处理相关的 deb 包是 untangle-libnfnetlink、untangle-libnetfilter-queue 和 untangle-libnetfilter-conntrack，与 kernel 下的 netfilter 补丁包对应。

pkgs 目录下是 Untangle 需要的一些软件包或后台程序的配置包，如 untangle-apache-config2 是 Web 服务器的配置和资源包；untangle-net-alpaca 是主要网络配置相关内容，包括系统 iptables、tc 的配置……每个 pkgs 里的软件包都相对独立，可以生成单独的 deb 安装包。

3. Untangle/src

src 目录下代码以 Java 为主，它是整个 Untangle 应用的主体。包含从网包获取和连接表的保存，Untangle 应用平台和应用组件。

网包获取和连接表代码涉及 libnetcap、libmvutil、libvector、jnetcap、jmvutil、jvector 几个目录。其中，libnetcap 中的代码主要是从网卡接收和发送网包，识别并保存会话，以代理的模式将一个会话拆分为两段，Untangle 本身作为一个中间代理，与连接的双方分别建立连接。libmvutil 和 libvector 将会话中的网包封装为自定义的事件对象(Event)，并通知已注册的组件(包含在一个观察者链表中)，令其对事件进行操作。jnetcap、jmvutil、jvector 分别将前三个包的函数实现与 Java 中的接口连接起来，通过 JNI 进行。

Untangle 应用框架 untangle-vm 用 Java 编写，主要包括 uvm-lib 和 uvm 两个目录。uvm-lib 目录为安全应用组件 node 依赖库和 Untangle 虚拟机的主程序入口。

Untangle 使用类似虚拟机的概念，提供 Untangle 安全应用组件需要的各种服务，包括各种管理器、管道等的接口和实现(如配置管理器、备份管理等)、应用组件的基类和接口等。

UVM 上的应用组件(Node)分为 FilterNode 和 Services Node。FilterNode 主要包括各种安全模块的实现,如防火墙组件,防病毒组件、垃圾邮件过滤组件。服务类模块有状态报告(report)、流量记录(sniffer)等。

21.1.2　UVM 代码结构

UVM 代码拆分为 uvm-lib 和 uvm 两个目录,uvm 目录里多为执行脚本,Java 代码则包含在 uvm-lib 目录中。

uvm-lib 代码结构如图 21-1 所示。

```
uvm-lib
  api
  bootstrap
  impl
  localapi
  servlets/alpaca/src
  servlets/library/src
  servlets/reports/src
  servlets/setup/src
  servlets/webui/src
  taglib
```

图 21-1　uvm-lib 代码结构

目录结构如下所示。

(1) api 目录提供 UVM 需要的接口和数据结构。

(2) localapi 目录提供 UVM 需要的接口和数据结构。

(3) impl 目录是 UVM 服务接口的实现,其主体为 com. untangle. uvm. engine 包里的类。

(4) bootstrap 目录内包含 UVM 启动和初始化,入口类为 Main. java 文件的基本类 com. untangle. uvm. engine. Main。

UVM 执行参数如下所示:

```
/usr/lib/jvm/java-6-sun/bin/java
-Djavax.net.ssl.trustStore=/usr/share/untangle/conf/cacerts
-Djava.endorsed.dirs=/usr/share/java/endorsed
-Djava.library.path=/usr/lib/uvm
-Dbunnicula.home=/usr/share/untangle
-Djdbc.drivers=org.postgresql.Driver
-Dcom.untangle.isDevel=false
-Djruby.shell=/bin/sh
-Dcom.untangle.cli.server.home=/usr/share/untangle-cli-server/src
-Xmx1600m -Xms128m
-XX:+UseCondMarkSweepGC -XX:+CMSIncrementalMode
-XX:+CMSIncrementalPacing -XX:CMSIncrementalDutyCycleMin=10
-XX:CMSIncrementalDutyCycle=50 -XX:CMSIncrementalSafetyFactor=90
-Xloggc:/var/log/uvm/gc.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps
-verbose:gc
-server -Xss96k
-Dje.maxMemory=120
-Dnetworkaddress.cache.ttl=30
-Dsun.net.inetaddr.ttl=30
-Dnetworkaddress.cache.negative.ttl=10
-Dlog4j.configuration=file:/usr/share/untangle/conf/log4j-uvm.xml
com.untangle.uvm.engine.Main
```

参数的配置为以下文件。

\untangle\untangle-src\uvm-lib\hier\etc\default\untangle-vm

```
# UVM server settings -*-sh-*-
```

```
source @PREFIX@/etc/default/untangle-activation

MAX_HEAP_SIZE=320m # to be overwritten by untangle-hw
MAX_BERKDB_MEG=32 # to be overwritten by untangle-hw
MAX_VIRTUAL_SIZE=1100000 # to be overwritten by untangle-hw

if [ -f @PREFIX@/etc/default/untangle-hw ] ; then
    source @PREFIX@/etc/default/untangle-hw # this one is dynamically generated by hw-config
fi

#
# untangle-vm configuration
#
UVM_ARGS=""

UVM_LOGDIR=@PREFIX@/var/log/uvm
UVM_RUNDIR=@PREFIX@/var/run
UVM_CONSOLE_LOG=${UVM_CONSOLE_LOG:-$UVM_LOGDIR/console.log}
UVM_WRAPPER_LOG=$UVM_LOGDIR/wrapper.log
UVM_GC_LOG=$UVM_LOGDIR/gc.log
MONIT_LOG=$UVM_LOGDIR/monit.log
PACKAGES_LOG=$UVM_LOGDIR/packages.log
UVM_CMD=@PREFIX@/usr/bin/uvm
UVM_USER=root
UVM_YJP=${UVM_YJP=$:-no}
UVM_NICENESS=0

## if in the build, disable network configuration by default
# UVM_ARGS="${UVM_ARGS} -Duvm.devel.nonetworking=true"

## Set a session limit-defaults to 10k
# UVM_ARGS="${UVM_ARGS} -Dargon.sessionlimit=${UVM_SESSION_LIMIT}"

## Enable the heap monitor.
# UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.enabled=true"

## Rate, kilobytes/second to start dumping stack traces.
# UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.rate=$((3 * 1024)"

## required minimum to start dumping memory (even if rate exceeds rate)
# UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.min=$(( 100 * 1024 ))"

## Level at which to just dump memory regardless of the rate
# UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.level=$(( 310 * 1024 ))"
```

```
## Poll frequencies in milliseconds
#UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.poll=500"

## Minimum number of milliseconds in between dumping two thread dumps
#UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.interval=3000"

## Name of the file where the log should go, if unspecified, this goes to console.log
#UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.memmonitor.file=heapmonitor.log"

## Lifetime for positive phonebook lookups (millis)
#UVM_ARGS="${UVM_ARGS} -Dcom.untangle.uvm.user.phonebook.lifetime=300000"

## Expiration time for host-bypassed sites (millis)
#UVM_ARGS="${UVM_ARGS} -Dcom.untangle.node.webfilter.bypass-timeout=3600000 -Dcom.untangle.node.
webfilter.bypass-sleep-delay=1200000"

## Limit Berkeley DB caching to 5% of java heap
UVM_ARGS="${UVM_ARGS} -Dje.maxMemory=${MAX_BERKDB_MEG}"

## Reduce DNS caching to minimal (30 seconds).
UVM_ARGS="${UVM_ARGS} -Dnetworkaddress.cache.ttl=30 -Dsun.net.inetaddr.ttl=30 -Dnetworkaddress.
cache.negative.ttl=10"

## Alternative store
#UVM_ARGS="${UVM_ARGS} -Duvm.store.url=https://store.untangle.com"

## Default Logging
UVM_ARGS="${UVM_ARGS}  -Dlog4j.configuration=file:@PREFIX@/usr/share/untangle/conf/log4j-uvm.xml"

# If inside the devel environment
if [ "x" != "x@PREFIX@" ] ; then
    ## Disable writing of networking config files
    UVM_ARGS="${UVM_ARGS} -Duvm.devel.nonetworking=true"
fi


#
# JAVA configuration
#
JAVA_HOME=@DEFAULT_JAVA6_HOME@
JAVA_OPTS=""
PATH=${JAVA_HOME}/bin:${PATH}

## Max Heap Size
JAVA_OPTS="${JAVA_OPTS} -Xmx${MAX_HEAP_SIZE}"
```

```
## Min Heap Size
JAVA_OPTS="${JAVA_OPTS} -Xms128m"

## Throughput Garbarge collector
#JAVA_OPTS="${JAVA_OPTS} -XX:+UseParallelGC -XX:GCTimeRatio=6"

## Concurrent Garbarge collector
JAVA_OPTS="${JAVA_OPTS} -XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode -XX:+CMSIncrementalPacing
-XX:CMSIncrementalDutyCycleMin=10 -XX:CMSIncrementalDutyCycle=50 -XX:CMSIncrementalSafetyFactor=90"

## Garbage collector
JAVA_OPTS="${JAVA_OPTS} -Xloggc:$UVM_GC_LOG -XX:+PrintGCDetails -XX:+
PrintGCDateStamps"

## Garbage collector verbosity
JAVA_OPTS="${JAVA_OPTS} -verbose:gc"

## Server mode (optimizes code at startup)
JAVA_OPTS="${JAVA_OPTS} -server"

if [ 'uname -m' != 'x86_64' ]; then
    ## Stack size
    JAVA_OPTS="${JAVA_OPTS} -Xss96k"
else
    ## Stack size
    JAVA_OPTS="${JAVA_OPTS} -Xss192k"
fi

# If inside the devel environment
if [ "x" != "x@PREFIX@" ] ; then
    ## Enable assertions
    JAVA_OPTS="${JAVA_OPTS} -ea"
fi

export JAVA_OPTS
export JAVA_HOME
export PATH


#
# OEM configuration
#
if [ -f @PREFIX@/etc/untangle/oem/oem.sh ] ; then
    source @PREFIX@/etc/untangle/oem/oem.sh
else
```

```
    OEM_NAME="Untangle"
fi
```

uvm-lib 中，localapi 目录下的 com. untangle. uvm. vnet. AbstractNode. java 是 UVM 所有应用组件主类的基类，其要求子类实现 getPipeSpecs 方法，每个 PipeSpec 都包含管道接入方式和会话事件处理类(基类为 com. untangle. uvm. vnet. AbstractEventHandler)。

21.1.3 Filter Node 代码结构

Filter Node 模块通过 getPipeSpecs 方法返回长度大于 0 的 PipeSpec 对象数组，实现对应数量的会话事件处理类。代表模块有 Firewall、Anti-Spyware、Protofilter 等。

Filter Node 模块代码目录主要包括 api、impl、hier。其中 api 和 impl 以 Java 代码实现。

api 目录中为应用组件需要用到的数据结构，以及对外接口的调用接口。重要的数据结构包括组件的配置类和事件类，它们通过 hibernate 持久化到数据库中(PostgreSQL)。api/resources/META-INF/annotated-classes 文件中，描述了该组件要持久化的类名。

impl 目录中为应用组件的具体实现，主要包括主类和会话事件处理类。包括介入网流会话处理的安全类模块和不介入网流处理的服务类模块。

hier 目录中为 untangle-vm 之外的系统功能文件如下。

(1) 为组件配置在数据库中对应表的创建 SQL 语句脚本。

```
src/<node-name>/hier/usr/share/untangle/schema/* .sql
```

(2) 生成每日报告的 python 脚本。

```
src/<node-name>/hier/usr/lib/python2.5/reports/node/* .py
```

(3) 网页管理配置页面的 js 脚本。

```
src\<node-name>\hier\usr\share\untangle\web\webui\script\untangle-node-<name>\settings.js
```

(4) Untangle 命令行管理工具 UCLI(Untangle Command Line Input)管理脚本。

```
src\<node-name>\hier\usr\share\untangle\nucli.rb
```

21.2 Untangle 开发环境

21.2.1 虚拟化系统开发调试环境

基于虚拟机开发调试环境可以免去物理主机安装的问题。测试 Untangle 系统，建议采用虚拟机的方法，配置虚拟化网络，在虚拟化网络中进行测试，避免物理安装的各种不足。以 VMware 公司 VMware Workstation 21.0 专业虚拟机软件为例，可以虚拟现有任何操作系统，而且使用简单，方便组建一个局域网内多台主机的实验。图 21-2 给出了 VMware 的虚拟网络配置图。

为了进行 Untangle 系统的开发，比较方便快捷的方式是采用虚拟网络。有必要了解一下虚拟机的虚拟网络配置。VMware 虚拟机主要包括虚拟机、真实主机以及其他的虚拟机进行通信。通信分两部分，一个是局域网内的，另一个是连接到公网的。VMware 虚拟机主

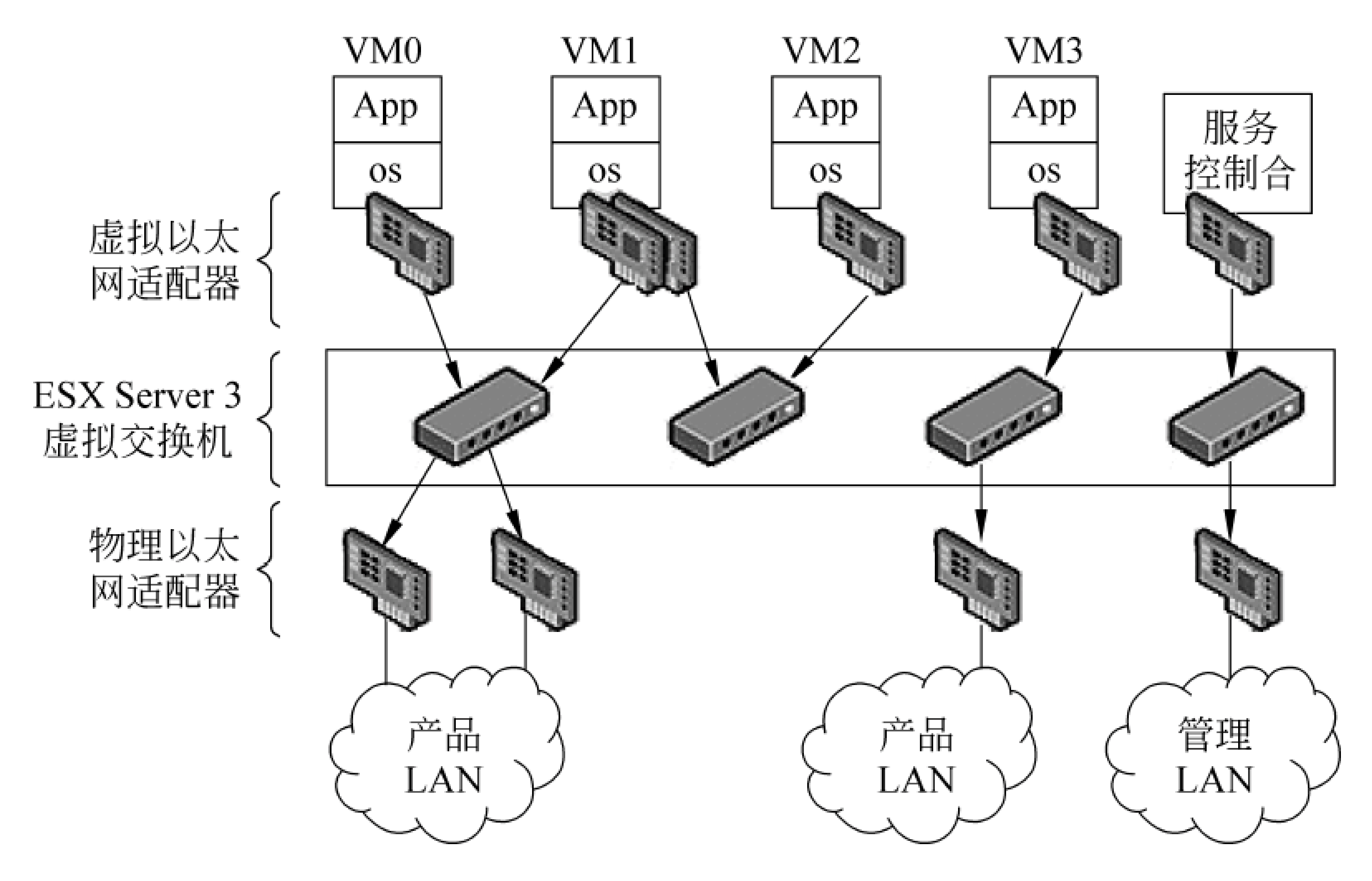


图 21-2　VMware 虚拟主机配置图

要有三种不同模式。

1. 直接桥接模式

直接桥接模式拓扑图如图 21-3 所示。

如果真实主机在一个以太网中，直接桥接模式是将虚拟机接入网络最简单的方法。虚拟机就像一个新增加的、与真实主机有着同等物理地位的一台计算机，桥接模式可以享受所有可用的服务，包括文件服务、打印服务等。

2. Host-only 模式

Host-only 模式拓扑图如图 21-4 所示。

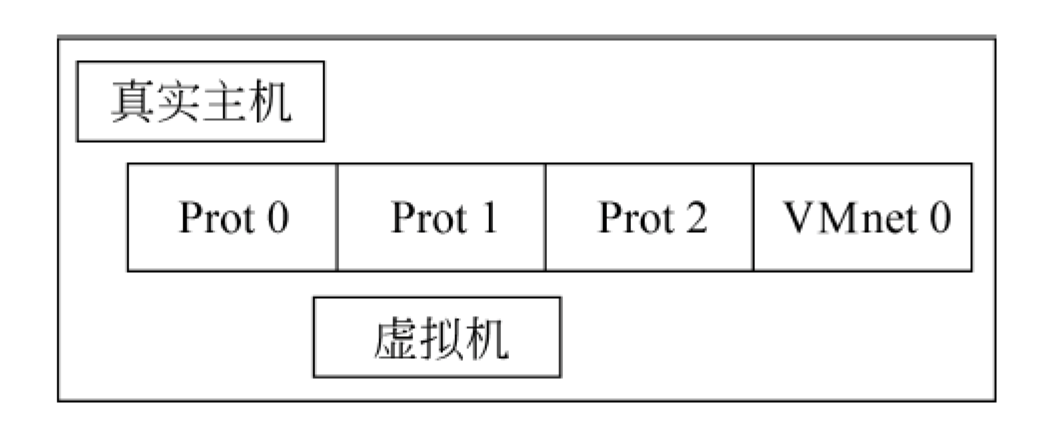


图 21-3　直接桥接模式拓扑图

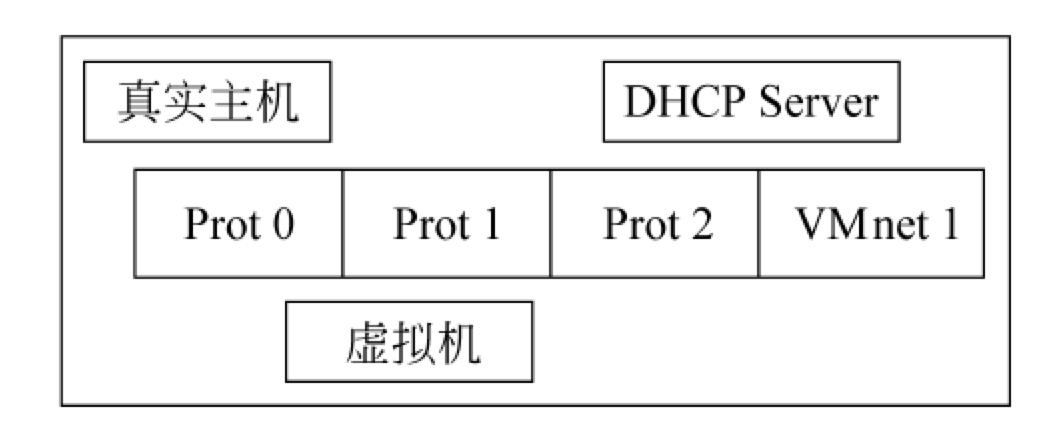


图 21-4　Host-only 模式拓扑图

Host-only 模式用来建立隔离的虚拟机环境，在这种模式下，虚拟机与真实主机通过虚拟私有网络进行连接，只有同为 Host-only 模式下的且在一个虚拟交换机（vSwitch）的连接下才可以互相访问，外界无法访问。Host-only 模式只能使用私有 IP 地址，IP 地址、网关，域名解析服务都由 VMnet 1 来分配。

3. NAT 模式

NAT 模式拓扑图如图 21-5 所示。

NAT(Network Address Translation)模式使虚拟机直接连接到公网最方便，缺点是桥接模式下的其他功能都不能享用。凡是选用 NAT 结构的虚拟机，均由 VMnet 8 提供 IP 地址、网关、域名解析。

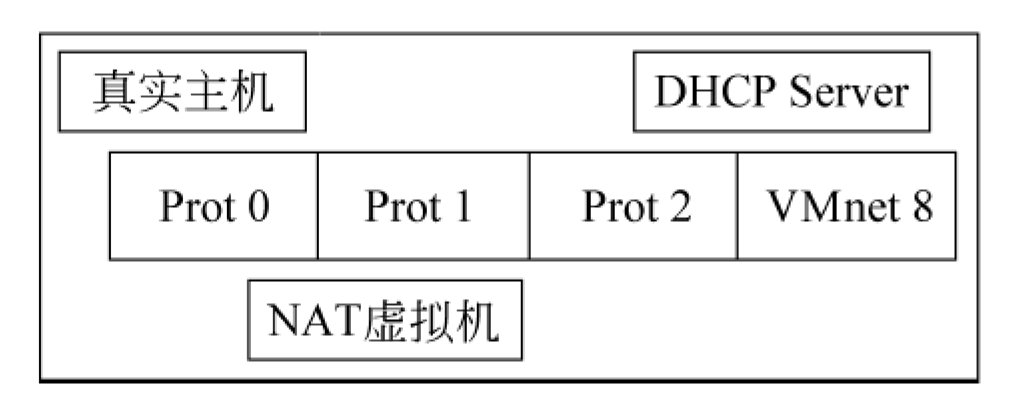


图 21-5　NAT 模式拓扑图

4. 虚拟网络设置

使用 VM 的高度可扩展网络模型组建非常复杂的局域网，可以进行 Untangle 在复杂网络环境下的测试。

图 21-6 所示为采用 VMware Workstation 自带的 Virtual Network Editor 进行虚拟机之间网络配置的例子。

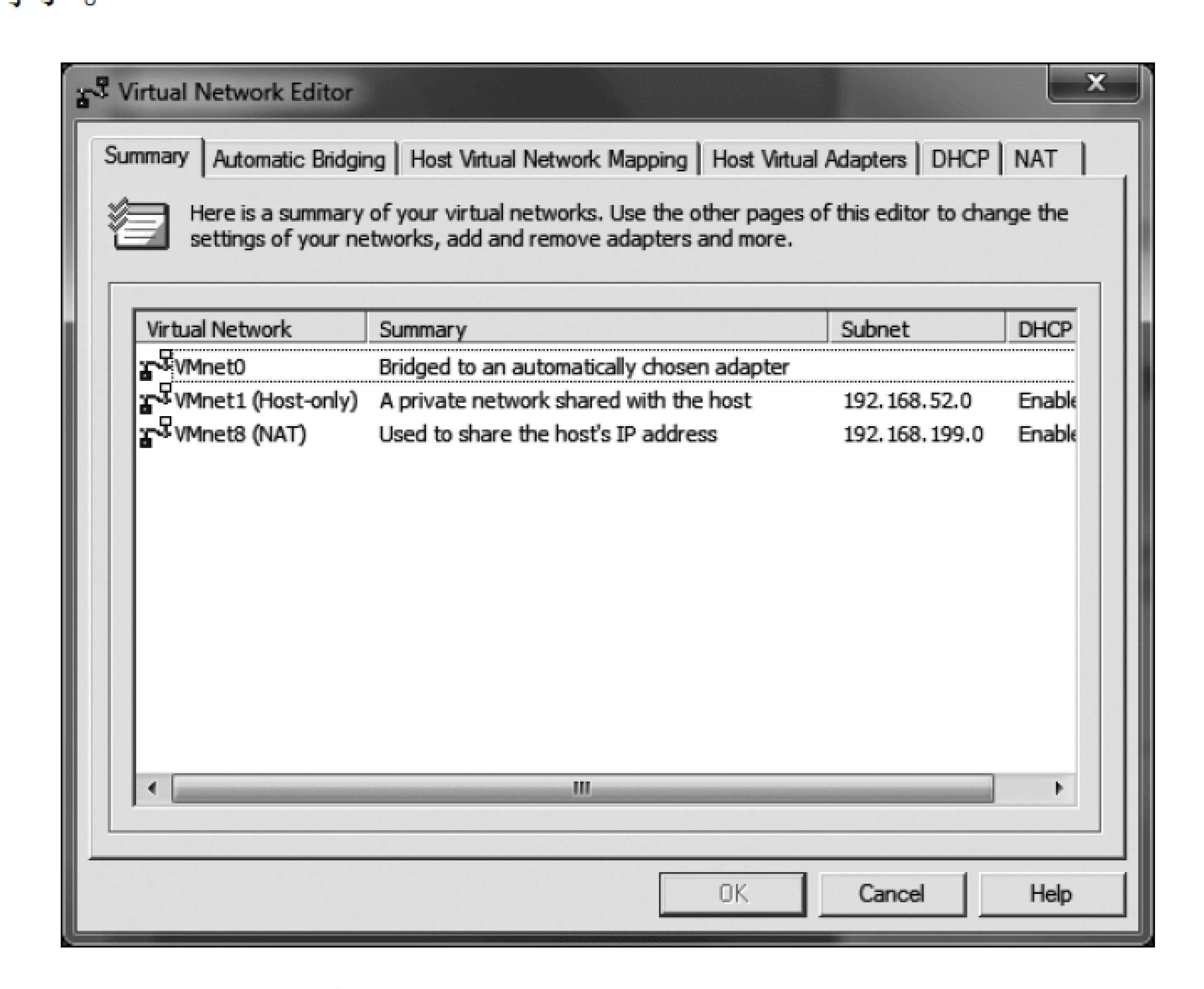


图 21-6　Virtual Network Editor

下面给出了基于 VMware 的 Untangle 机器和两台 Ubuntu 机器的连接的拓扑实例，定义了 VMnet 8 用于外网的 NAT 模式，当然也可以直接采用桥接模式，模拟 Untangle 机器的外网环境，VMnet 2 用于 Untangle 机器的内网环境，Ubuntu 机器的 IP 地址都是从 Untangle 机器进行分配的。

典型的 Untangle 虚拟开发环境如图 21-7 所示。

21.2.2　Untangle 源代码开发环境

源代码开发环境主要是代码阅读和代码修改。可以采用 SourceInsight、Eclipse、UltraEdit、EditPlus、Emacs、Vim 等。这里推荐使用 SourceInsight，综合能力比较强，适合整个代码树的阅读。尽管在查找方面不如前面 UltraEdit 好用。Eclipse 对于 Java 代码阅读与开发比较适用。

在 Eclipse 中建立 Untangle 的开发环境，Eclipse 可采用 Indigo、Helios 和 Galileo。

(1) 把 Eclipse 的 workspace 选择到 src 目录。

(2) 打开工程的 Properties 窗口。

(3) 建立 uvm-lib 的 Java Project，把以下目录加入为 Source(可选：在每个 Source

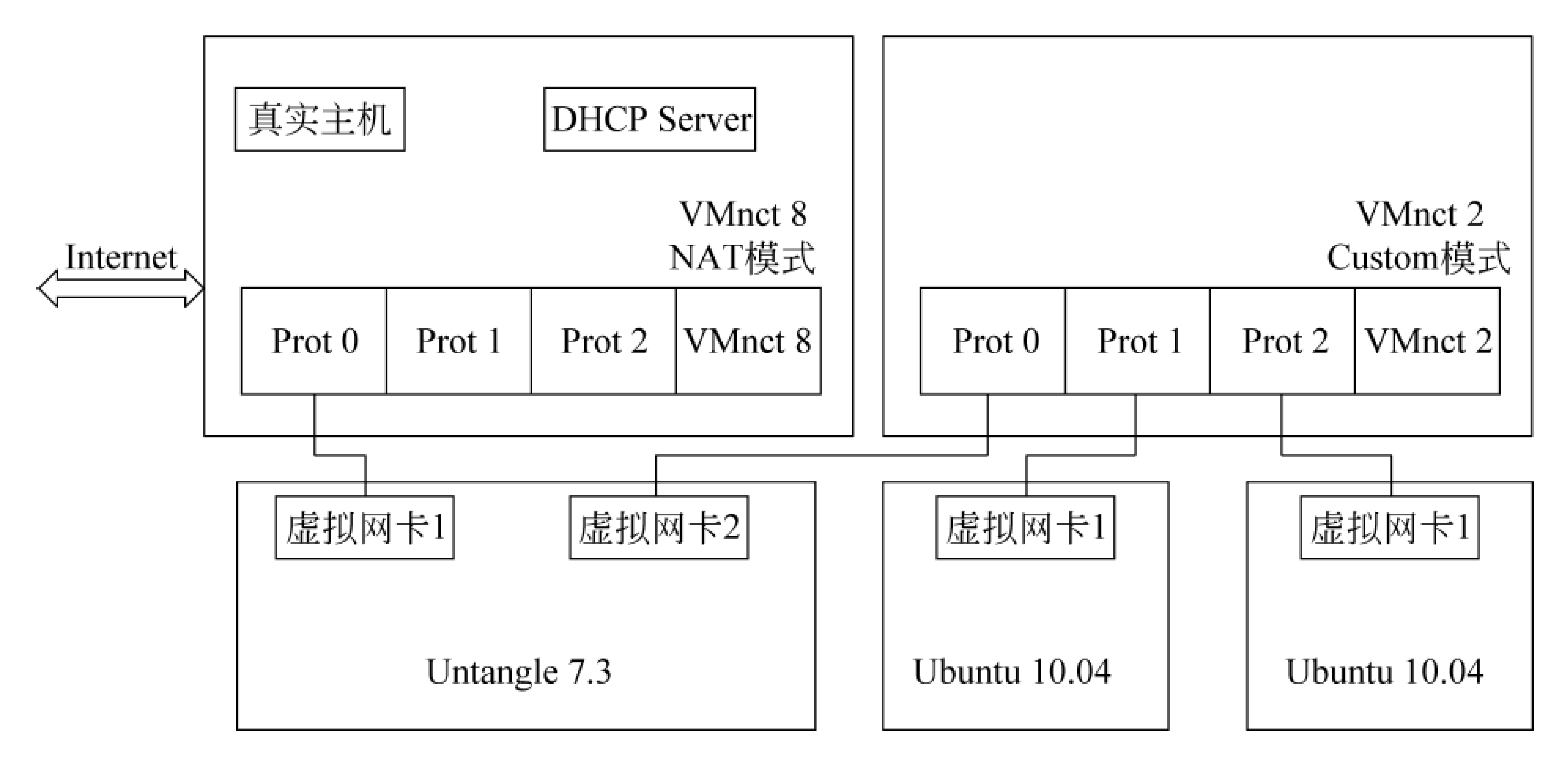


图 21-7　Untangle 虚拟开发环境

Folder 的 Exclude 加入 * */.svn/* ,把 svn 的文件排除在外,使编译时不会把 svn 文件复制进 bin 目录),如图 21-8 所示。

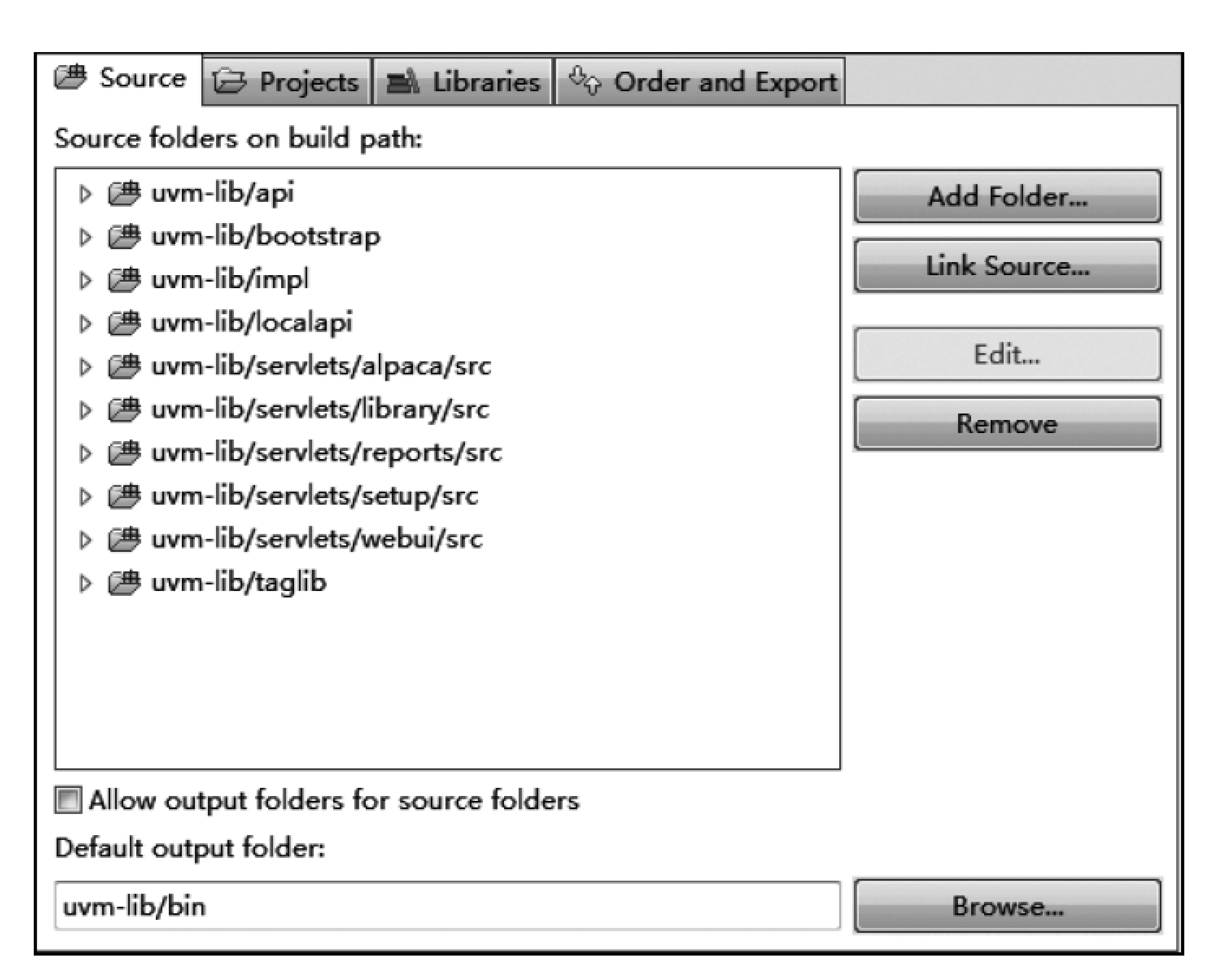


图 21-8　Source 选项卡

(4) 在 Libraries 选项卡中选择刚刚建立的 uvm,把 uvm-lib/lib/里的所有 jar 添加到其中,连续单击“确认”按钮。

(5) uvm-lib 工程应该就没有红叉了。

(6) 建立其他 java project,如 Firewall。

(7) 把 Java 工程中的 api 和 impl 加入为 Source Folder(如前)。

(8) 在 Libraries 选项卡中,单击 Add Library 按钮,选择刚刚建立的 uvm,如图 21-9 所示。

(9) 在 Library 选项卡中,单击 Add Class Folder 按钮,选择 uvm-lib/bin 即可,如图 21-10 所示。

某些 node XX Project 还需要依赖其他工程的编译结果,例如:

clam-base 依赖 virus-base/bin。

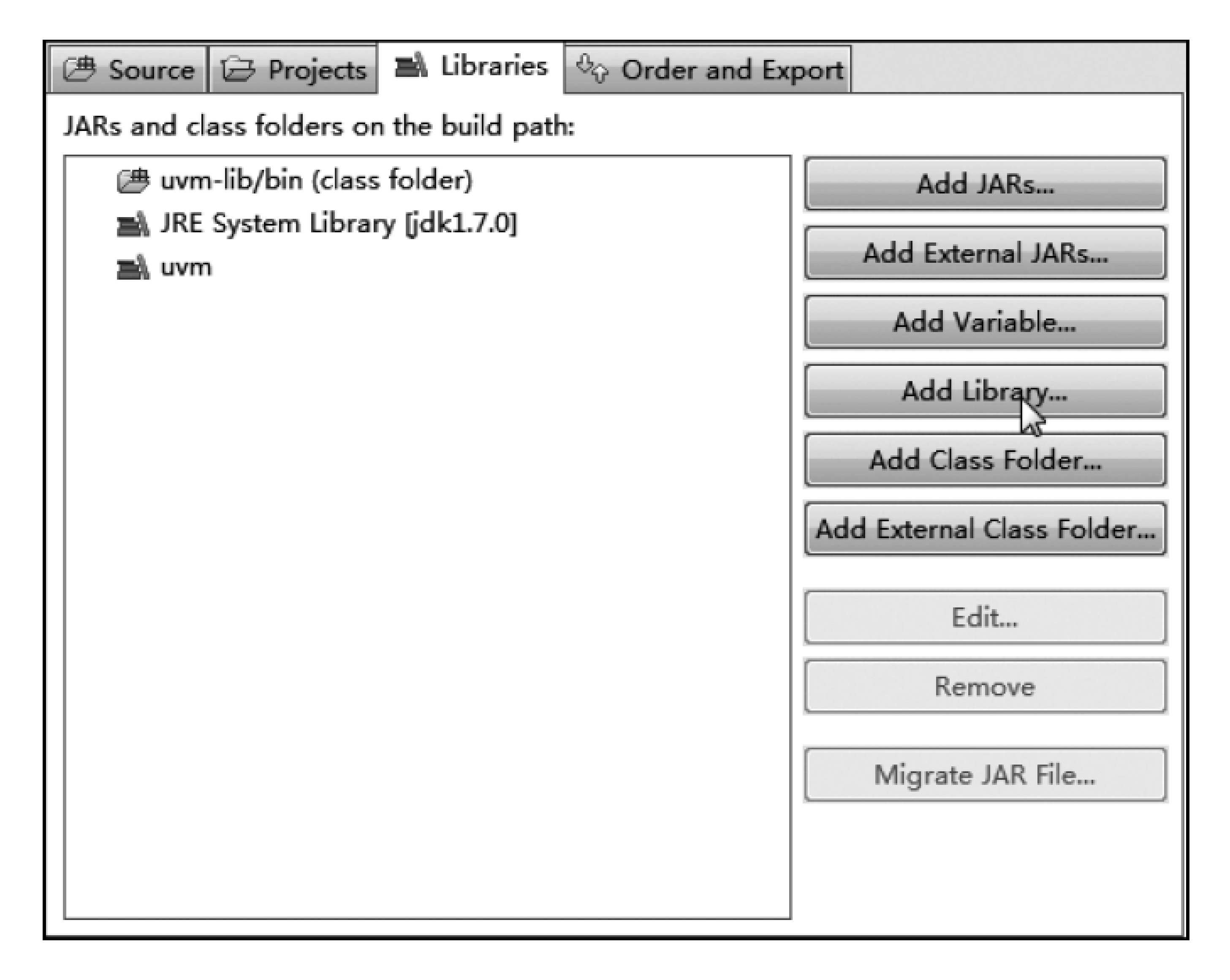


图 21-9　Libraries 选项卡

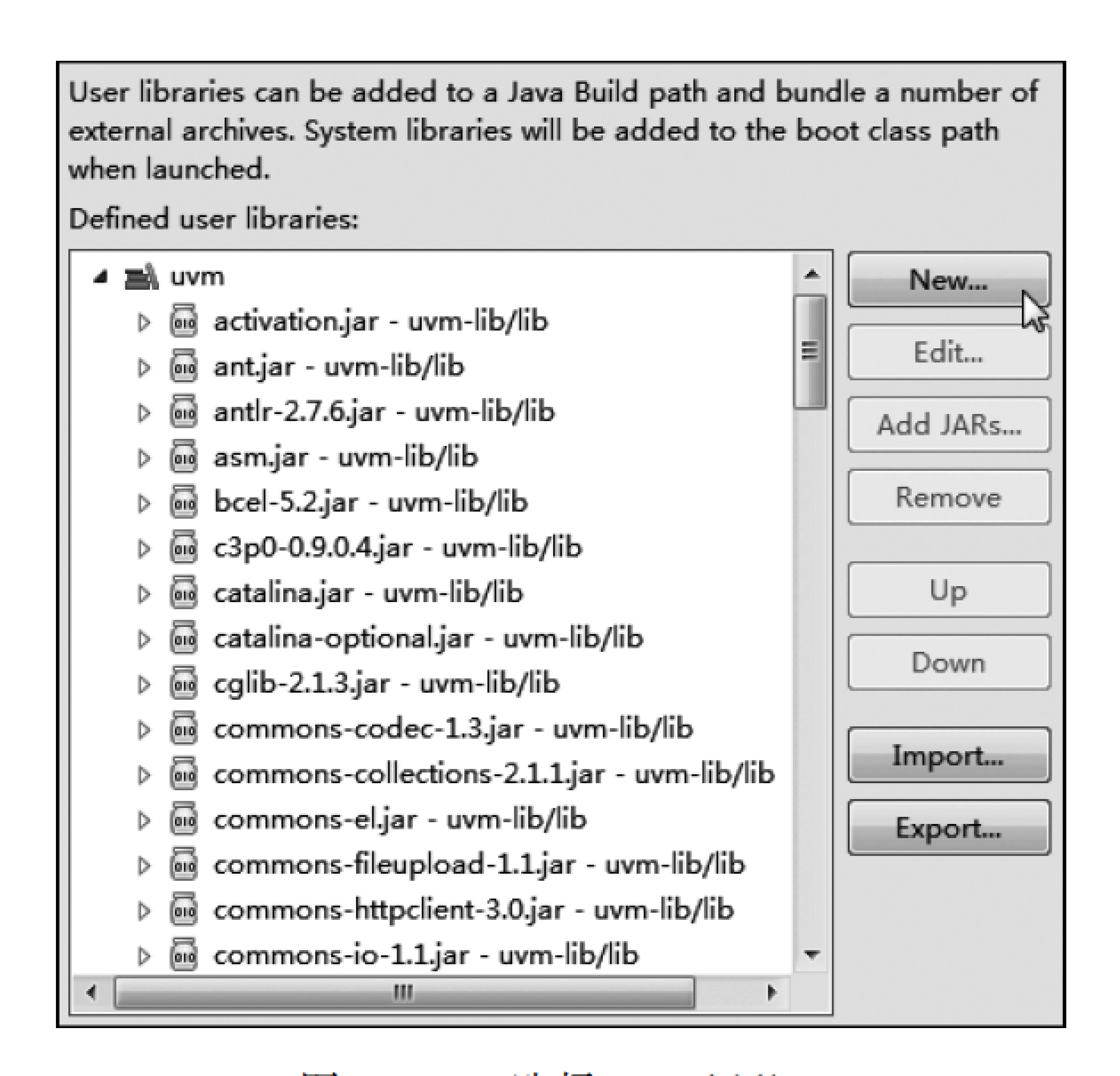


图 21-10　选择 urm-liblbin

clam 依赖 virus-base/bin 和 clam-base/bin。

phish 依赖 spamassassin/bin、spam-base/bin 和 clam-base/bin。

spamassassin 依赖 spam/bin。

webfilter 依赖 webfilter-base/bin。

21.3　Untangle 源代码编译链接框架

21.3.1　Untangle 源代码编译环境

参见 wiki build the code。

（1）安装 Untangle 系统。

（2）使用默认密码 passwd 登录 http://localhost/alpaca，配置外部接口。
（3）启用 ssh（删除/etc/ssh/sshd_not_to_be_run，运行'$ /etc/init.d/ssh start'）。
（4）通过终端登录，ssh 连接 root@ip.addr。
（5）删除某些 Untangle 核心包。

```
$ apt-get --purge --yes remove untangle-vm
```

（6）重配置 debconf，选择 Dialog 作为接口，其他设置为默认值。

```
$ dpkg-reconfigure debconf
```

（7）安装 untangle-development 包。

```
$ apt-get update
$ apt-get --yes install untangle-development-build
```

（8）获取代码。
源代码在 svn://svn.untangle.com/trunk 下面。

```
$ svn cosvn://svn.untangle.com/trunk
```

最新的源代码在 http:// gitorious.org/untangle 下面。命名自己的 clone 版本为 src-yours，使用以下命令生成自己命名的自有版本：

```
$ git clone git@gitorious.org:untangle/src-yours.git
```

（9）贡献代码和修改。
使用 Gitorious 接口，生成 merge 请求。
（10）匿名 cloning。
匿名 cloning 代码库（用于测试）采用以下命令：

```
$ git clone git://gitorious.org/untangle/src.git
```

21.3.2 Untangle 源代码编译方法

编译在 Untangle 源码包 src 目录下进行，src 目录下有如下内容。
（1）buildtools/：系统编译时用到的脚本和库。
（2）rakefile：主要 ruby makefile 文件。
（3）<component>-name：每个 Node、Casing 和 Component 的目录。
（4）prj.el：Emacs JDEE 项目文件。
（5）G*：GNU 全局标签文件（Global Tag Files）。
编译过程生成如下产物。
（1）dist/：开发编译的结果，安装系统的镜像。
（2）staging/：编译生成的类和其他预编译的产物 artifacts。
（3）taskstamps.txt：编译目标的时戳文件。
编译分为直接编译和软件包编译两种。

1. 直接编译

在 src 目录运行 rake,Untangle 将被编译到 src/dist 里,dist 中包含 etc、var、usr、tmp 等目录,与根目录结构一致,可在 src 目录通过运行./dist/etc/init.d/untangle-vm start 来启动 Untangle 服务。

实际编译过程相对简单,进入到需要编译的 src 目录:

```
$ rake
```

如果编译成功,在开发编译生成的系统将安装在 dist 目录,直接启动 untangle-vm 即可。

```
$ sudo ./dist/etc/init.d/untangle-vm start
```

此方法较快捷,适用于开发阶段。

若编译中提示找不到包,可以尝试运行 export SRC_HOME=.,再进行编译。

2. 软件包编译

在 src 目录运行 debuild -us -uc -d,将 Untangle 各软件包编译为 deb 包,可用于定形后的软件包生成。

```
$ apt-get build-dep untangle-vm
$ debuild -us -uc
```

生成的 deb 包将出现在上级目录(..)中,可用以下命令进行安装:

```
$ dpkg -i ../untangle-*deb
```

21.3.3 Untangle 源代码编译过程

源代码 untangle/src/目录编译采用 rake(ruby make)文件来指导编译过程。rake 替换了 Linux make 系统,untangle/src/rakefile 相当于 makefile 文件的作用。

主要调用了 untangle/src/buildtools/*.rb,包括 buildtools.rb、rake-util.rb、node.rb、untangle-core.rb。辅助的 rb 文件还有 stamp-task.rb、c-compiler.rb、jars.rb、target.rb。

顺序是 rakefile→buildtools.rb→node.rb→untangle-core.rb。

其中 buildtools.rb 中检查 src/downloads/,如果不是 cleanbuild,将重新编译 src/downloads/*.tar 文件,并转移到/usr/share/java/uvm/中。

调用 rake 命令过程中,会执行每个 node 下(如 untangle/src/clam)的 package.rb 写的编译文件,调用 untangle/src/buildtools/node.rb 中的函数库。

node.rb 中对每个 node 下的文件夹,按照如下顺序 localapi→impl→api→po→hier 进行处理。

编译过程中,生成 untangle/src/staging,其中的 grabbag 用于存储 jar 文件,include 目录用来存放头文件。

以\src\buildtools\untangle-core.rb 文件为例说明:

```
##Require all of the sub packages.
##Done manually because order matters.
```

```
## XXX Could create a new helper method that sets a prefix directory before
## calling require and then unsets it afterwards.
require "#{SRC_HOME}/libmvutil/package.rb"
require "#{SRC_HOME}/libnetcap/package.rb"
require "#{SRC_HOME}/libvector/package.rb"
require "#{SRC_HOME}/jmvutil/package.rb"
require "#{SRC_HOME}/jnetcap/package.rb"
require "#{SRC_HOME}/jvector/package.rb"
require "#{SRC_HOME}/nfutil/package.rb"
require "#{SRC_HOME}/uvm/package.rb"
require "#{SRC_HOME}/uvm-lib/package.rb"

require "#{SRC_HOME}/test/package.rb"
require "#{SRC_HOME}/reporting/package.rb"
require "#{SRC_HOME}/ftp-casing/package.rb"
require "#{SRC_HOME}/http-casing/package.rb"
require "#{SRC_HOME}/mail-casing/package.rb"
require "#{SRC_HOME}/spyware/package.rb"
require "#{SRC_HOME}/router/package.rb"

require "#{SRC_HOME}/shield/package.rb"
require "#{SRC_HOME}/firewall/package.rb"
require "#{SRC_HOME}/openvpn/package.rb"
require "#{SRC_HOME}/protofilter/package.rb"
require "#{SRC_HOME}/sigma/package.rb"
require "#{SRC_HOME}/ips/package.rb"

## Base Nodes
require "#{SRC_HOME}/spam-base/package.rb"
require "#{SRC_HOME}/virus-base/package.rb"
require "#{SRC_HOME}/clam-base/package.rb"
require "#{SRC_HOME}/webfilter-base/package.rb"

## SPAM based nodes
require "#{SRC_HOME}/phish/package.rb"
require "#{SRC_HOME}/spamassassin/package.rb"

## Webfilter based nodes
require "#{SRC_HOME}/webfilter/package.rb"

# Ad Blocker node
require "#{SRC_HOME}/adblocker/package.rb"

## Virus based nodes
require "#{SRC_HOME}/clam/package.rb"
```

```
##Other packages
require "#{SRC_HOME}/util/package.rb"

##sniffer node
require "#{SRC_HOME}/sniffer/package.rb"

##collaborator node
require "#{SRC_HOME}/collaborator/package.rb"

##softprotect node
require "#{SRC_HOME}/softprotect/package.rb"

##softbypass node
require "#{SRC_HOME}/softbypass/package.rb"

##watchdog node
require "#{SRC_HOME}/watchdog/package.rb"

wlibs=[]

libuvmcore_so="#{BuildEnv::SRC.staging}/libuvmcore.so"

archives=['libmvutil', 'libnetcap', 'libvector', 'jmvutil', 'jnetcap', 'jvector']

##Make the so dependent on each archive
archives.each do |n|
  file libuvmcore_so =>BuildEnv::SRC[n]['archive']
end

file libuvmcore_so do
  compilerEnv=CCompilerEnv.new({"flags"=>"-pthread
#{CCompilerEnv.
defaultDebugFlags}"})
  archivesFiles=archives.map {|n| BuildEnv::SRC[n]['archive'].filename }

  CBuilder.new(BuildEnv::SRC, compilerEnv).makeSharedLibrary(archivesFiles, libuvmcore_so,
  [],
                              ['xml2', 'sysfs', 'netfilter_queue','netfilter_conntrack'], wlibs)
end

BuildEnv::SRC['untangle-libuvm']['impl'].register_dependency(libuvmcore_so)

BuildEnv::SRC.installTarget.install_files(libuvmcore_so,
"#{BuildEnv::SRC['untangle-libuvmcore'].distDirectory}/usr/lib/uvm")
```

Untangle 中的 6 个 C 代码库文件为 libnetcap、libmvutil、libvector 和 jnetcap、jmvutil、

jvector，最后编译为 libuvmcore. so。

deb 包编译过程由 debian control 文件确定，见/untangle/src/debian/control 文件：

```
Source: untangle-vm
Section: net
Priority: optional
Maintainer: Sebastien Delafond <seb@untangle.com>
Uploaders: Untangle Buildbot <buildbot@untangle.com>
Build-Depends: untangle-development-build
Standards-Version: 3.8.0

Package: untangle-libuvmcore
Architecture: i386 amd64
Depends: ${misc:Depends}, libc6 (>=2.3.2), untangle-libnetfilter-queue0 (>=0.0.15),
libsysfs1 | libsysfs2, libnetfilter-conntrack1 (>=0.0.81) | untangle-libnetfilter-
conntrack1 (>=0.0.81)
XB-Display-Name: Untangle Core Library
XB-Untangle-Pkg-Type: library
XB-Desc-Icon:  XXXXXX
Description: Untangle Core Library
 Untangle Core Library files.

Package: untangle-casing-ftp
Architecture: all
Depends: untangle-libuvm
XB-Display-Name: File Transfer Casing
XB-Untangle-Pkg-Type: casing
XB-Desc-Icon: XXXXXX
Description: The Untangle File Transfer/FTP Decoder
 The Untangle File Transfer/FTP Decoder casing.

Package: untangle-casing-http
Architecture: all
Depends: untangle-libuvm
XB-Display-Name: Web Casing
XB-Untangle-Pkg-Type: casing
XB-Desc-Icon: XXXXXX
Description: The Untangle Web Decoder
 The Untangle Web Decoder casing.

Package: untangle-casing-mail
Architecture: all
Depends: untangle-libuvm
XB-Display-Name: Email Casing
XB-Untangle-Pkg-Type: casing
```

XB-Desc-Icon: XXXXXX
Description: The Untangle Email Decoder
 The Untangle Email Decoder casing.

Package: untangle-node-shield
Architecture: all
Depends: untangle-vm, untangle-shield, dash
XB-Display-Name: Attack Blocker
XB-Untangle-Pkg-Type: service
XB-View-Position: 1020
XB-Desc-Icon: XXXXXX
Description: Attack Blocker
 The Attack Blocker application.

Package: untangle-node-spyware
Architecture: all
Depends: untangle-vm, untangle-casing-http
XB-Display-Name: Spyware Blocker
XB-View-Position: 20
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Spyware Blocker
 The Spyware Blocker application.

Package: untangle-node-webfilter
Architecture: all
Depends: untangle-vm, untangle-casing-http, untangle-base-webfilter, untangle-webfilter-init
XB-Display-Name: Web Filter
XB-View-Position: 30
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Web Filter
 The Web Filter application.

Package: untangle-node-clam
Architecture: all
Depends: untangle-vm, untangle-base-virus, untangle-clamav-config, untangle-casing-http, untangle-casing-ftp, untangle-casing-mail
XB-Display-Name: Virus Blocker
XB-View-Position: 70
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Virus Blocker
 The Virus Blocker application.

```
Package: untangle-node-protofilter
Architecture: all
Depends: untangle-vm
XB-Display-Name: Protocol Control
XB-View-Position: 80
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Protocol Control
 The Protocol Control application.

Package: untangle-node-router
Architecture: all
Depends: untangle-vm, dnsmasq, untangle-casing-ftp
XB-Display-Name: Router
XB-View-Position: 1000
XB-Auto-Start: true
XB-Invisible: true
XB-Untangle-Pkg-Type: service
XB-Desc-Icon: XXXXXX
Description: Router
 The Router application.

Package: untangle-node-firewall
Architecture: all
Depends: untangle-vm
XB-Display-Name: Firewall
XB-View-Position: 90
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Firewall
 The Firewall application.

Package: untangle-node-spamassassin
Architecture: all
Depends: untangle-vm, untangle-base-spam, untangle-casing-mail
XB-Display-Name: Spam Blocker
XB-View-Position: 15
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Spam Blocker
 The Spam Blocker application.

Package: untangle-node-phish
Architecture: all
Depends: untangle-vm, untangle-base-spam, untangle-clamav-config,
```

```
untangle-casing-mail, untangle-casing-http
XB-Display-Name: Phish Blocker
XB-View-Position: 17
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Phish Blocker
 The Phish Blocker application.

Package: untangle-node-openvpn
Architecture: all
Depends: untangle-vm, untangle-nsis-addons, openssl, openvpn
XB-Display-Name: OpenVPN
XB-View-Position: 1019
XB-Untangle-Pkg-Type: service
XB-Desc-Icon: XXXXXX
Description: OpenVPN
 The OpenVPN application.

Package: untangle-node-ips
Architecture: all
Depends: untangle-vm, untangle-casing-http, untangle-snort-rules
XB-Display-Name: Intrusion Prevention
XB-View-Position: 75
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
Description: Intrusion Prevention
 The Intrusion Prevention application.

Package: untangle-node-reporting
Architecture: all
Depends: untangle-vm
XB-Display-Name: Reports
XB-View-Position: 1022
XB-Untangle-Pkg-Type: service
XB-Desc-Icon: XXXXXX
Description: Reports
 The Reports application.

Package: untangle-node-adblocker
Architecture: all
Depends: untangle-vm, untangle-casing-http
XB-Display-Name: Ad Blocker
XB-View-Position: 120
XB-Untangle-Pkg-Type: node
XB-Desc-Icon: XXXXXX
```

```
Description: Ad Blocker
 The Ad Blocker application.

Package: untangle-node-cpd
Architecture: all
Depends: untangle-vm, untangle-cpd, php5-pgsql, php5-curl
XB-Display-Name: Captive Portal
XB-View-Position: 31
XB-Untangle-Pkg-Type: service
XB-Desc-Icon: XXXXXX
Description: Captive Portal
 The Captive Portal.


Package: untangle-base-virus
Architecture: all
Depends: untangle-libuvm, untangle-casing-http, untangle-casing-mail
XB-Display-Name: Virus Blocker Base
XB-Untangle-Pkg-Type: base
XB-Desc-Icon: XXXXXX
Description: Virus Blocker Base
 The Virus Blocker Base.

Package: untangle-base-spam
Architecture: all
Depends: untangle-vm, untangle-spamassassin-update, untangle-casing-mail
XB-Display-Name: Spam Blocker Base
XB-Untangle-Pkg-Type: base
XB-Desc-Icon: XXXXXX
Description: Spam Blocker Base
 The Spam Blocker Base.

Package: untangle-base-webfilter
Architecture: all
Depends: untangle-libuvm, untangle-casing-http
XB-Display-Name: WebFilter Base
XB-Untangle-Pkg-Type: base
XB-Desc-Icon: XXXXXX
Description: WebFilter Base
 The WebFilter Base.

Package: untangle-libuvmthirdparty
Architecture: all
Depends: sun-java6-jre
Conflicts: mvvm (<<5.0), libuvm-thirdparty, libmvvm-thirdparty
```

Provides: libuvm-thirdparty
Replaces: libuvm-thirdparty
XB-Display-Name: Platform Thirdparty Libraries
XB-Untangle-Pkg-Type: library
XB-Desc-Icon: XXXXXX
Description: Platform Thirdparty Libraries
 The Platform Thirdparty Libraries.

Package: untangle-vm
Architecture: all
Depends: untangle-keyring, sun-java6-jdk, untangle-libuvm, untangle-libuvmcore, untangle-libuvmthirdparty, sudo, iptables (>=1.3.6), db4.2-util, untangle-ujac2pdf, at, ebtables, mime-support, untangle-nsis-addons, dnsmasq, pppoe, bridge-utils, iputils-ping, curl, wget, sg3-utils, pciutils, libdbd-pg-ruby, ruby1.8, ruby, untangle-linux-image, exim4-daemon-light | mail-transport-agent, gettext, libgettext-ruby, ethtool, net-tools, dnsutils, untangle-net-alpaca, untangle-node-router, untangle-ldap-server, libgpgme-ruby, dash, untangle-apache2-config (>>6.2), python-pycurl | python2.3-pycurl, libwww-mechanize-ruby, perl, untangle-vm-shell, zip, unzip, tofrodos, python-psycopg, python-egenix-mxdatetime, python-lxml, tidy, dpkg-dev, python-codespeak-lib, python-reportlab, python-imaging, untangle-hardware-config, untangle-monit-config, untangle-arp-eater, untangle-php-config
Recommends: untangle-postgresql-config, untangle-gateway-light
Provides: mvvm
Conflicts: mvvm, untangle-restore-tools (<<5.1), untangle-linux-image-2.6.16-ck11-untangle-prod-486, untangle-linux-image-2.6.16-ck11-untangle-prod-xd-486, untangle-linux-image-2.6.16-ck11-untangle-cd-486, untangle-slapd, untangle-libldap-2.2-7, untangle-ldap-utils, untangle-client
Replaces: mvvm
XB-Display-Name: Platform
XB-Untangle-Pkg-Type: library
XB-Desc-Icon: XXXXXX
Description: Platform
 The Platform.

Package: untangle-libuvm
Architecture: all
Conflicts: untangle-vm (<<5.1)
Depends: sun-java6-jre
XB-Display-Name: Platform Libraries
XB-Untangle-Pkg-Type: library
XB-Desc-Icon: XXXXXX
Description: Platform Libraries
 The Platform Libraries.

```
Package: untangle-buildutil
Architecture: all
Depends: sun-java6-jre
Description: Platform Build Utils
 The Untangle Build Utils.
```

develbuild 结果放在 untangle/src/dist/目录下面。

21.4 Untangle 安全应用 Node 开发

参见 Untangle wiki 的 Application Developers Guide。

21.4.1 安全应用 Node 相关技术

本节为在 Untangle 平台上编写新的安全应用(也称为 Node，节点)提供指导和说明。

在 Untangle 平台上提供新的安全应用，可以快捷地进行部署。本节主要的用例采用 Ad Blocker。Ad Blocker 代码可以在 svn 服务器上获取。

编写新的安全应用需要熟悉以下技术，包括 Linux、debian、Java、json、extjs 等。

21.4.2 安全应用 Node 分类

节点或应用有两种类型：Filter Node 和 Service Node。

Filter Node 位于 rack 机架上方，透明处理网络流量。多个 Filter Node 可以实例化在一个 Untangle 服务器中(每机架)，不同的 policy 可以基于不同 Filter Node 实现。Ad Blocker 和 Web Filter 都是 Filter Node 的例子。

Service Node 位于 rack 机架下方，是全局的。一个 Untangle 服务器只有一种。OpenVPN 和 Remote Access Portal 都是 Service Node 的例子。

安全应用的选择取决于选择不同的类型。一般来说，如果是处理网络流量的应用，用 Filter Node 来实现。如果应用是全局的或提供单一服务或者代表 Server 上已经运行的服务，可以用全局 Service Node。

21.4.3 安全应用 Node 源代码说明

在编写一个新的 Node 时，需要了解 Node 的整个生命周期的架构和含义，以 Ad Blocker 为例。Ad Blocker 源代码在/src/adblocker 目录下。

以下主要以 Ad Blocker 为例，解释基本开发知识。Service Node 与之差别甚少。

编译系统设计主要目的是为了帮助以标准的布局来创建组件 Node。典型的 src Node 位于目录 src/<node_name>(如 src/adblocker)，并可能包含如下的基本单元。

(1) package.rb：ruby make 的编译文件，用于 rake 自动编译该 Node。

(2) localapi：仅用于本地客户端的 api，untangle-casing-http、untangle-casing-mail、untangle-casing-ftp node 和 libuvm 目录有。对于绝大多数节点，一般不需要。

(3) api：用于远程或本地客户端的 api。

(4) impl：主要实现代码。

(5) hier：用于 linux 文件目录，该文件夹包含节点的 db 文件、UI 显示和 report 文件。某些文件通过 buildtools/rake-util.rb 的 filterset 进行 filtered。

(6) servlets：包含与该组件相关的 serverlet。如对应 WebFilter blocking pages 的 servlet 就在该目录下。

(7) po：po/pot 文件包含用于 I18N 的字符串键值。

(8) unittest：单元测试文件 unittest。

21.4.4 安全应用 Node 编译与打包

为了编译，新的节点应该在完整的 src 目录树下，编译系统主要用于编译节点使之遵循标准的布局。

Untangle 编译系统有两种工作模式。

(1) 在 dist 目录，开发者编译整个源代码系统，这样可以在开发环境中直接运行开发版本，验证开发功能是否有效。

(2) deb 包编译用来生成可以用 apt-get/dpkg 安装的 Debian 包。

开发者可以通过以上两种方法，对创建的安全应用 Node 进行调试和测试。

1. 编译文件

package.rb 代码负责编译相应的 Node，主要有两个 ruby 类。

(1) NodeBuilder：该 ruby 类用于编译 Node 源代码

(2) ServletBuilder：该 ruby 类用于编译与 Node 相关的 Servlets 源代码。

以/src/AdBlocker 为例，其 packge.rb 文件如下：

```
NodeBuilder.makeNode(BuildEnv::SRC, 'untangle-node-adblocker', 'adblocker',
                     [BuildEnv::SRC['untangle-casing-http']['localapi']])
```

以/src/phish 为例，其 packge.rb 文件如下：

```
mail=BuildEnv::SRC['untangle-casing-mail']
http=BuildEnv::SRC['untangle-casing-http']
spam=BuildEnv::SRC['untangle-base-spam']
clam=BuildEnv::SRC['untangle-base-clam']
phish=BuildEnv::SRC['untangle-node-phish']

NodeBuilder.makeNode(BuildEnv::SRC, 'untangle-node-phish', 'phish',
                     [mail['localapi'], http['localapi']],
                     [], { 'spam-base' =>spam, 'clam-base' =>clam })

deps=[http['localapi'], phish['impl'], spam['impl']]

ServletBuilder.new(phish, 'com.untangle.node.phish.jsp',
                   "./phish/servlets/idblocker", [],
                   deps, [], [BuildEnv::SERVLET_COMMON])
```

其中，NodeBuilder.makeNode 函数参数解释如下。

第一个参数 BuildEnv::SRC，用于引用编译环境。

第二个参数，用于定义节点的名字。

第三个参数，用于指定在 src 目录下 adblocker 的位置

最后参数是可选的，用来说明该节点的依赖性。在 adblocker 组件下，与之相关的是 HTTP network stream。对于 adblocker，依赖于 http casing node 来处理 HTTP 流。对于其他节点，此项为空。

为了让编译系统编译该 package，应该在 buildtools/untangle-core.rb 的文件加入引用。由于 Ad Blocker node 依赖于 untangle-casing-http node。Ad Blocker node 应该在 HTTP Casing 之后编译。因此，在 buildtools/untangle-core.rb 文件中，加入：

```
# Ad Blocker node
require "#{SRC_HOME}/adblocker/package.rb"
```

2. 安全应用 Node 打包

debian/control 文件位于 src/debian/control。对每个节点，需要在 node 的 deb 包的 control fields 中加入相应字段。除了标准的 control fields (Package、Architecture、Depends、Conflicts、Provides、Replaces、Description etc)，一组 Untangle 特有的 control fields 也应该填写(用前缀 XB 表示)。

XB-Untangle-Pkg-Type：包类型可为 node、casing、service、library、base、lib _ item、trial、unknown。

XB-Display-Name：node 的显示名称。

XB-View-Position：视图位置(Apps 和 Rack 使用)。

XB-Auto-Start：指定是否安装后即刻自动启动。

XB-Invisible：指定 node 是否可见。

XB-Desc-Icon：node/package icon 图标文件，其中 icon 是 42×42 的 png 文件。

使用 base64 命令来创建 base64 编码。可采用如下命令：

```
# 'cat icon.png | base64'
```

21.4.5 安全应用 Node API

Node API 应该提供公有的运行时控制方法来操作实例的状态，可用性(启用/关闭)(Node Interface 的一部分)和配置 Node Settings 的方法(Node Interface 的一部分)。

```
package com.untangle.node.adblocker;

import java.util.List;

import com.untangle.uvm.logging.EventManager;
import com.untangle.uvm.node.Node;
import com.untangle.uvm.node.StringRule;

public interface AdBlocker extends Node {
    /* *
     * Allows the user interface to get a set of
```

```
<code>EventRepository</code>s and <code>LogEvent<code>s
     *
     * @return the <code>EventManager</code>of the node
     */
    EventManager<AdBlockerEvent>getEventManager();

    /**
     * @return the base settings for the Ad Blocker node
     */
    AdBlockerBaseSettings getBaseSettings();

    /**
     * Sets the base settings for the Ad Blocker node
     *
     * @param baseSettings
     */
    void setBaseSettings(AdBlockerBaseSettings baseSettings);

    /**
     * Getter for the rules
     *
     * @param start index
     * @param limit number of return elements
     * @param sortColumns column names that match Hibernate's idea (basically the 'bean
       property name'),
     * prefixed by "+" or "-" to specified the sorting order: ascending respective
       descending.
     *
     * @return list of filtered and sorted rules
     */
    List<StringRule>getRules(int start, int limit, String... sortColumns);

    /**
     * Setter for the rules
     *
     * @param added list of added rules
     * @param deleted list of deleted rules specified by theirs ids
     * @param modified list of modified rules
     */
    void updateRules(List<StringRule>added, List<Long>deleted,
                     List<StringRule>modified);

    /**
     * Update all settings (base settings and lists) once, in a single transaction
     *
```

```
    * @ param baseSettings base settings for the Ad Blocker node
    * @ param rules- list containing three lists for added, deleted and
      modified rules
    * /
   void updateAll(AdBlockerBaseSettings baseSettings, List[] rules);

}
```

1. 安全应用 NodeSettings API

安全应用 NodeSettings API 包括如下内容。

1）NodeSettings

NodeSettings API 属于 Node interface。

NodeSettings API 在 NodeImpl class 类中实现(如 AdBlockerImpl)。

每个 node 有一个新的 BaseSettings class（即 AdBlockerBaseSettings),包含 node 的所有设置。唯一的不同之处是 BaseSettings class 只保存每个实际 list 的长度,而非实际 list 本身。

节点的属性表格的填写至多两次操作完成：其中一次获取 BaseSettings,另一次获取当前 tab 的每一项。

2）Lists

有两类 list：static 和 pageable。

static lists 包含如下内容

（1）ordered lists（如 firewall rules、policy rules)。

（2）unordered lists（如即将更新的 packages,这些链表不会进一步增长)dynamic lists 为其他类型的链表

3）Static Lists

用 accessor 来获取整个 list。

对于 ordered list,返回的 list 也是有序的。

对于 unordered list,返回的 list 也是无序的,必须由 client 来排序,用 settor 来设置整个 list。

4）Dynamic Lists

由 hibernate 从数据库中实际取得,由服务器端的 server API 进行排序。

Dynamic Lists 的 accessor 如下：

```
List<StringRule>getRules(int start, int limit, String... sortColumns);
```

参数包括：

起始索引——start index;

limit——返回 elements 的限制数目;

sortColumns——列名匹配 Hibernate 的 idea(即“bean 属性名”),前缀上＋或－来指定排序的顺序：升序或者降序。

Dynamic Lists 的 settor 如下：

```
void updateRules(List<StringRule> added, List<Long> deleted, List<StringRule> modified);
```

参数包括：

增加的 element 由 client 端创建，具有 null 型 Id。

删除的 elements 的 Id 由它原有的 Id 值来指定。

修改的 elements 不改变 Id。

为了优化，base settings 和 list 的更新可由一次 transaction 来完成，updateAll 方法定义如下：

```
void updateAll(AdBlockerBaseSettings baseSettings, List[] rules);
```

参数包括：

Ad Blocker node 的 base settings；

对于每个动态 list，包括 added、deleted 和 modified elements 3 个链表 Events 事件；

Events API 允许用户 interface 来获取一组 EventRepositorys 和每个 EventRepository 对应的 LogEvents。

```
EventManager<AdBlockerEvent>getEventManager();
```

2. Resources 文件

文件夹 api/resources 包含 node 的资源文件。一个 node 有两种类型的 resource 文件，位于 api/resources/META-INF 目录下。

(1) annotated-classes：包含 hibernate annotated api classes。

(2) uvm-node. xml：包含 node 配置信息。

(3) node-desc：定义 node 实例的属性，使用以下 node-desc 单元。

① single-instance：仅允许单个 node 实例。

② classname：该 node 的实现类(如 com. untangle. node. adblocker. AdBlockerImpl)。

③ node-base：如果 node 是某个基类 node 的继承，则该属性标识该基类 node。

④ power-button：指定安装的 node 是否有开关按钮。

⑤ no-start：指定安装的 node 是否需要启动。

(4) parent：用于说明 node 之间的依赖关系(如 casing 依赖)。

(5) uvm-resource：指定输出给 UVM 的 jar 文件。

21.4.6 安全应用 Node 实现

位于＜nodePackageName＞/impl 文件夹中，提供运行时控制方法的实现（操作实例的状态）和配置节点的 settings 方法。

```
public class AdBlockerImpl extends AbstractNode implements AdBlocker
```

1. 安全应用 Node 实现过程

1) 设置的加载和保存

NodeImpl class 中应该有一份缓存的设置实例：

```
private AdBlockerSettings settings=null;
```

初始设置由 initializeSettings 方法来创建和保存，该方法在 node 创建时调用。

```
public void initializeSettings() {
    AdBlockerSettings settings=new AdBlockerSettings(getTid());

    logger.info("Loading Filters...");
    Set<StringRule>ruleSet=RulesLoader.loadRules();
    settings.setRules(ruleSet);

    setAdBlockerSettings(settings);
    logger.info(ruleSet.size()+" filters loaded");
}
```

这些设置应该通过 postInit method 从数据库中加载。在节点启动之前，该方法在节点实例创建时调用。

```
protected void postInit(String[] args) throws NodeException {
    logger.info("Post init");
    queryDBForSettings();

    reconfigure();
}
```

postInit 是 Node 生命周期中的一个方法。关于生命周期方法的更多细节，请参考 Node 的 Interface 类和 AbstractNode 类

采用 TransactionWork 来执行所有数据库操作，TransactionWork 封装了一个工作单元在 Hibernate transaction 中执行。

```
private void queryDBForSettings() {
    TransactionWork tw=new TransactionWork()
        {
            public boolean doWork(Session s)
            {
                 Query q= s.createQuery("from AdBlockerSettings abs where abs.tid= :
                 tid");
                q.setParameter("tid", getTid());
                AdBlockerImpl.this.settings=
                (AdBlockerSettings)q.uniqueResult();
                return true;
            }

            public Object getResult() { return null; }
        };
    getNodeContext().runTransaction(tw);
}
```

通过访问实例的缓存设置来获取设置和基本设置（以 AdBlockerSettings 和 AdBlockerBaseSettings 为例）：

```
public AdBlockerBaseSettings getBaseSettings() {
    return settings.getBaseSettings();
}
...
AdBlockerSettings getSettings() {
    return settings;
}
```

更新设置和基本设置之前，应该先在数据库中持久化改变，并且更新相应的缓存设置实例：

```
public void setBaseSettings(final AdBlockerBaseSettings baseSettings) {
    TransactionWork tw= new TransactionWork() {
        public boolean doWork(Session s) {
            settings.setBaseSettings(baseSettings);
            settings= (AdBlockerSettings)s.merge(settings);
            return true;
        }

        public Object getResult() {
            return null;
        }
    };
    getNodeContext().runTransaction(tw);
}
⋮
private void setAdBlockerSettings(final AdBlockerSettings settings) {
    TransactionWork tw= new TransactionWork()
        {
            public boolean doWork(Session s)
            {
                AdBlockerImpl.this.settings=
                (AdBlockerSettings)s.merge (settings);
                return true;
            }

            public Object getResult() { return null; }
        };
    getNodeContext().runTransaction(tw);

    reconfigure();
}
```

获取/更新 Node 的 lists，可由 PartialListUtil 的 helper class 来完成，如下所示。

```
private final PartialListUtil listUtil=new PartialListUtil();
```

获取绝大多数的 dynamic lists，可由 PartialListUtil. getItems()的 helper 方法来完成，如下所示：

```
public List<StringRule>getRules(int start, int limit, String... sortColumns) {
    return listUtil.getItems(
            "select hbs.rules from AdBlockerSettings hbs where hbs.tid = ? :tid ",
            getNodeContext(), getTid(), start, limit, sortColumns);
}
```

更新 node 的 lists，可由 PartialListUtil. updateCachedItems()的 helper 方法来完成。这将更新缓存设置，其中包括 list 的改变，接着将缓存设置持久化到数据库中。

```
@Override
public void updateRules(List<StringRule>added, List<Long>deleted,
        List<StringRule>modified) {
    updateRules(settings.getRules(), added, deleted, modified);
}

private void updateRules(final Set rules, final List added,
        final List<Long>deleted, final List modified) {
    TransactionWork tw=new TransactionWork() {
        public boolean doWork(Session s) {
            listUtil.updateCachedItems(rules, added, deleted, modified);

            settings= (AdBlockerSettings) s.merge(settings);

            return true;
        }

        public Object getResult() {
            return null;
        }
    };
    getNodeContext().runTransaction(tw);
}
```

为了能够从客户的单次请求来更新所有设置，需要在单次 transaction 中实现更新所有设置的方法。

```
public void updateAll ( final AdBlockerBaseSettings baseSettings, final List [ ]
rulesModifications) {
    TransactionWork tw=new TransactionWork() {
        public boolean doWork(Session s) {
            if (baseSettings !=null) {
                settings.setBaseSettings(baseSettings);
```

```
                }

                listUtil.updateCachedItems(settings.getRules(),
                rulesModifications);
                settings= (AdBlockerSettings)s.merge(settings);

                return true;
            }

            public Object getResult() {
                return null;
            }
        };
        getNodeContext().runTransaction(tw);

        reconfigure();
    }
```

2）统计

每个 Node 可以有两组 Blingers 集合：Activity Blinger（左侧）图形化 node 的活跃性和 Counter Blinger（右侧）来显示 node 行为的统计信息。

为了定义 Node Blingers，开发者需要为当前 Node 实例获取 LocalMessageManager 和计数值：

```
LocalMessageManager lmm=
LocalUvmContextFactory.context().localMessageManager();
Counters c= lmm.getCounters(getTid());
```

添加所需的 Blingers：

```
scanBlinger= c.addActivity("scan", I18nUtil.marktr("Pages scanned"), null, I18nUtil.
marktr("SCAN"));
blockBlinger= c.addActivity("block", I18nUtil.marktr("Ads blocked"), null, I18nUtil.
marktr("BLOCK"));
```

第一个参数表示 Blinger 的名称。

第二个参数表示 Activity Blinger（Blinger 左侧）。

第三个参数表示 Blinger 计数值的测量值。

最后参数表示 Counter Blinger 的显示名称（Blinger 右侧）；如果不需要，则设为 null。

由于一次只能显示 4 个 Blingers 数值，激活的 Blinger 数据仅显示一次，默认的定义如下：

```
lmm.setActiveMetricsIfNotSet(getTid(), scanBlinger, blockBlinger);
```

3）日志事件和过滤事件

为该 Node 创建 EventManager，即 eventLogger 类实例。

```
    eventLogger=
```

```
EventLoggerFactory.factory().getEventLogger(getNodeContext());
```

创建 Filter Events 事件为过滤合适的日志事件(见 AdBlockedFilter 来过滤 AdBlockerEvent blocked events)并注册该过滤器到 event manager:

```
AdBlockedFilter eventFilter= new AdBlockedFilter();
eventLogger.addSimpleEventFilter(eventFilter);
```

创建作为 NodeImpl 部分的方法来记录过滤事件,从 handler 中调用 logging event 方法来记录对应的事件:

```
void log(AdBlockerEvent event) {
    eventLogger.log(event);
}
```

4) 网络流的处理

此步为安全功能实现的核心。网络流的处理通过插入 UVM pipes 机制来访问指定的网络数据,如以 HTTP 流量处理为例。

```
private final PipeSpec[] pipeSpecs;
⋮
public AdBlockerImpl() {
⋮
     SoloPipeSpec httpPipeSpec= new SoloPipeSpec("adblocker- http", this,
     new TokenAdaptor(this, new AdBlockerFactory(this)), Fitting.HTTP_TOKENS,
     Affinity.SERVER, 0);
     pipeSpecs= new PipeSpec[] { httpPipeSpec };
⋮
}
// AbstractNode methods - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - -
@ Override
protected PipeSpec[] getPipeSpecs() {
     return pipeSpecs;
}
⋮
```

其中:

(1) PipeSpec 有两种:SoloPipeSpec 和 CasingPipeSpec,普通 Node 采用 SoloPipeSpec。只有 FTP、HTTP 和 Mail 处理需要用 CasingPipeSpec。

(2) Fitting.HTTP_TOKENS→http tokens。

(3) Affinity:表示特定侧管道的亲和性。

① client:更快处理。

② server:后续处理。

(4) AdBlockerFactory:创建 AdBlockerHandlers 的 Factory。

(5) AdBlockerHandler:AdBlockerHandlers 实现实际的 HTTP 流量处理功能。

① 适配 HTTP token 流给予 protocol 状态相关的处理方法。

② 屏蔽不欢迎的 HTTP 流量：如不受欢迎的 Ad URLs。

(6) AdBlockerReplacementGenerator：生成被阻止 HTTP 流量的一个替代网页。

2. 数据库 schema 文件

数据库 schema 文件位于：

```
hier/usr/share/untangle/schema/untangle- node- <node_ name>
```

以 Adblocker 为例：

```
hier/usr/share/untangle/schema/untangle- node- adblocker
```

该文件夹包含的 script 文件 settings-schema. sql 和 events-schema. sql 来支持创建 table schema，创建 node 数据库的 node setting 和 node events 模板内容。

如果需要，还包括不同版本之间的更新脚本，如 settings-convert-x. sql 和 events-convert-x. sql。

21.4.7 安全应用 Node UI 界面

1. 为新 Node 创建 settings. js 文件

节点的设置文件必须以 settings. js 命名。设置文件必须在以下目录：

```
<nodePackageName>/hier/usr/share/untangle/web/webui/script/<node- name>
```

以 Adblocker 为例：

```
adblocker/hier/usr/share/untangle/web/webui/script/untangle- node- adblocker/settings.js
```

2. 防止重复加载 settings. js 文件

setting. js 文件为动态加载。为了防止重复加载，可按如下代码来编写：

```
if(!Ung.hasResource["Ung.AdBlocker"]) {
Ung.hasResource["Ung.AdBlocker"]=true;
    //JavaScript code...
}
```

3. 注册 Node(JavaScript)class

设置 Settings class 名称必须先注册如下：

```
Ung.NodeWin.registerClassName('untangle- node- adblocker','Ung.AdBlocker');
```

4. 创建 Node JavaScript class

Node 设置 Settings 类必须由 Ung. NodeWin 组件来扩展：

```
Ung.AdBlocker=Ext.extend(Ung.NodeWin, {
    //JavaScript code...
}
```

5. 重写 initComponent 方法

用相应的标签页重写 initComponent 方法来创建标签页面板。

```
initComponent?: function() {
```

```
        this.buildStatus();
        this.buildFilters();
        this.buildEventLog();
        // builds the tab panel with the tabs
        this.buildTabPanel([this.panelStatus, this.gridFilters, this.gridEventLog]);
        Ung.AdBlocker.superclass.initComponent.call(this);
    },
```

6. 创建 Grid Editor panel

以下例子给出了 Ung. EditorGrid 的使用：

```
// Filters grid
buildFilters?: function() {
    // enable is a check column
    var liveColumn= new Ext.grid.CheckColumn({
        header?: "<b>"+ this.i18n._("enable")+ "</b>",
        dataIndex : 'live',
        fixed : true
    });
    this.gridFilters= new Ung.EditorGrid({
        settingsCmp : this,
        name : 'Filters',
        helpSource : 'filters',
        // the total records is set from the base settings
        // rulesLength field
        totalRecords : this.getBaseSettings().rulesLength,
        emptyRow : {
            "string" : this.i18n._("[no description]"),
            "live" : true
        },
        title : this.i18n._("Filters"),
        // the column is autoexpanded if the grid width permits
        autoExpandColumn : 'string',
        recordJavaClass : "com.untangle.uvm.node.StringRule",
        // this is the function used by Ung.RpcProxy to retrive data
        // from the server
        proxyRpcFn : this.getRpcNode().getRules,
        // the list of fields
        fields : [{
            name : 'id'
        // this field is internationalized so a converter was
        // added
        }, {
            name : 'string',
            type : 'string'
        }, {
```

```
            name : 'live'
        }],
        // the list of columns for the column model
        columns : [liveColumn,
                {
            id : 'string',
            header : this.i18n._("description"),
            width : 200,
            dataIndex : 'string',
            // this is a simple text editor
            editor : new Ext.form.TextField({
                allowBlank : false
            })
        }],
        sortField : 'string',
        columnsDefaultSortable : true,
        plugins : [liveColumn],
        // the row input lines used by the row editor Window
        rowEditorInputLines : [
            new Ext.form.Checkbox({
            name : "Enable",
            dataIndex : "live",
            fieldLabel : this.i18n._("Enable")
        }), new Ext.form.TextField({
            name : "Description",
            dataIndex : "string",
            fieldLabel : this.i18n._("Description"),
            width : 200
        })]
    });
}
```

7. 创建 Grid Events Log panel

以下例子给出了 Ung. GridEventLog 的使用：

```
// Event Log
buildEventLog? : function() {
    var asClient= function(value) {
        var pe= (value ==null?null: value.pipelineEndpoints);
        return pe ===null?"": pe.CClientAddr+ ":"+ pe.CClientPort;
    };
    var asServer= function(value) {
        var pe= (value ==null ?null : value.pipelineEndpoints);
        return pe ===null ?"" : pe.SServerAddr+ ":"+ pe.SServerPort;
    };
    var asRequest= function(value) {
```

```
        return value == null || value.url == null ? "" : value.url;
    }.createDelegate(this);

    this.gridEventLog = new Ung.GridEventLog({
        settingsCmp : this,
        //the list of fields
        fields : [{
            name : 'timeStamp',
            sortType : Ung.SortTypes.asTimestamp //a custom "casting" function used
            to convert node values before sorting
        }, {
            name : 'displayAction',
            mapping : 'actionType',
            type : 'string', // for string sorting
            convert : function(value) {
                switch (value) {
                    case 0 : // PASSED
                        return this.i18n._("pass");
                    default :
                    case 1 : // BLOCKED
                        return this.i18n._("block");
                }
            }.createDelegate(this)
        }, {
            name : 'reason',
            type : 'string'
        }, {
            name : 'client',
            mapping : 'requestLine',
            sortType : asClient
        }, {
            name : 'server',
            mapping : 'requestLine',
            sortType : asServer
        }, {
            name : 'request',
            mapping : 'requestLine',
            sortType : asRequest
        }],
        //the list of columns
        columns : [{
            header : this.i18n._("timestamp"),
            width : 120,
            sortable : true,
            dataIndex : 'timeStamp',
```

```
                renderer : function(value) {
                    return i18n.timestampFormat(value);
                }
            }, {
                header : this.i18n._("action"),
                width : 120,
                sortable : true,
                dataIndex : 'displayAction'
            }, {
                header : this.i18n._("client"),
                width : 120,
                sortable : true,
                dataIndex : 'client',
                renderer : asClient
            }, {
                header : this.i18n._("request"),
                width : 120,
                sortable : true,
                dataIndex : 'request',
                renderer : asRequest
            }, {
                header : this.i18n._("reason for action"),
                width : 150,
                sortable : true,
                dataIndex : 'reason'
            }, {
                header : this.i18n._("server"),
                width : 120,
                sortable: true,
                dataIndex: 'server',
                renderer: asServer
            }]
        });
    },
```

8. 创建自定义面板 custom panels

以下例子给出了如何创建自定义面板 custom panel：

```
//Status Panel
buildStatus? : function() {
    this.panelStatus= new Ext.Panel({
        name : 'Status',
        helpSource : 'status',
        parentId : this.getId(),

        title : this.i18n._('Status'),
```

```
            layout : "form",
            cls: 'ung-panel',
            autoScroll : true,
            defaults : {
                xtype : 'fieldset',
                autoHeight : true,
                buttonAlign : 'left'
            },
            items : [{
                title : this.i18n._('Statistics'),
                layout:'form',
                labelWidth: 230,
                defaults: {
                    xtype: "textfield",
                    disabled: true
                },
                items: [{
                    fieldLabel : this.i18n._('Total Filters Available'),
                    name: 'Total Filters Available',
                    value: this.getBaseSettings().totalAvailable
                }, {
                    fieldLabel : this.i18n._('Total Filters Enabled'),
                    name: 'Total Filters Enabled',
                    value: this.getBaseSettings().totalBlocking
                }]
            }, {
                title : this.i18n._('Note'),
                cls: 'description',
                html : String.format(this.i18n._("{0} continues to maintain the default
                filters through automatic updates. You are free to modify and add filters,
                however it is not required."),
                            main.getBrandingBaseSettings().companyName)
            }]
        });
    }
```

21.4.8 Validation

1. Grid/row 编辑器的域验证

1）使用 validators

```
new Ext.form.TextField({
    name: "string",
    fieldLabel: this.i18n._("Site"),
    width: 200,
    //this filed is required
```

```
    allowBlank: false,
    //change default blankText error msg
    blankText: this.i18n._("Invalid \"URL\" specified"),
    //validate field data; the same mechanism is used for row editor and grid editor fields
    validator: function(fieldValue) {
        if (fieldValue.indexOf("https") == 0) {
            return this.i18n._("\"URL\" specified cannot be passed because it uses secure http
             (https)");
        }
        if (fieldValue.indexOf("http://") == 0) {
            fieldValue= fieldValue.substr(7);
        }
        if (fieldValue.indexOf("www.") == 0) {
            fieldValue= fieldValue.substr(4);
        }
        if (fieldValue.indexOf("/") == fieldValue.length- 1) {
            fieldValue= fieldValue.substring(0,
            fieldValue.length- 1);
        }
        if (fieldValue.trim().length == 0) {
            return this.i18n._("Invalid \"URL\" specified");
        }
        return true;
    }.createDelegate(this);
})
```

2）使用 ExtJS vTypes

```
new Ext.form.TextField({
    allowBlank?: false,
    vtype?: 'ipAddress'
})
```

2．Global validation（保存之前）

客户端 validation：

```
validateClient: function() {
    //no need for validation here...just alter the URLs
    if(this.gridPassedUrls) {
        this.alterUrls(this.gridPassedUrls.getSaveList());
    }
    if(this.gridBlockedUrls) {
        this.alterUrls(this.gridBlockedUrls.getSaveList());
    }
    return true;
},
```

服务端 validation:

```
validateServer: function() {
    //ipMaddr list must be validated server side
    var
passedClientsSaveList=this.gridPassedClients?this.gridPassedClients.
getSaveList():null;
    if(passedClientsSaveList!=null) {
        var ipMaddrList=[];
        //added
        for(var i=0;i<passedClientsSaveList[0].list.length;i++) {
ipMaddrList.push(passedClientsSaveList[0].list[i]["ipMaddr"]);
        }
        //modified
        for(var i=0;i<passedClientsSaveList[2].list.length;i++) {
ipMaddrList.push(passedClientsSaveList[2].list[i]["ipMaddr"]);
        }
        if(ipMaddrList.length>0) {
            try {
                var result=
this.getValidator().validate({list:
ipMaddrList, "javaClass":"java.util.ArrayList"});
                                if(!result.valid) {
                            this.panelPassLists.onManagePassedClients();
                            this.gridPassedClients.focusFirstChangedDataByFieldValue
                            ("ipMaddr",result.cause);
                            Ext.MessageBox.alert(this.i18n._("Validation
                            failed"), this.i18n._(result.message)+": "+result.cause);
                                  return false;
                }
            } catch (e){
                Ext.MessageBox.alert(i18n._("Failed"),e.message);
                                    return false;
            }
        }
    }
    return true;
},
```

3. 告警设置修改丢失

加入告警,当取消保存时,设置修改将被丢失。

```
isDirty: function() {
    return this.gridFilters.isDirty(); // just call isDirty for the grid
}
```

4. 创建保存设置的方法

获取修改后的数据,用 JSON-RPC 调用相应的 API 函数。

```
// save function
saveAction: function() {
    Ext.MessageBox.wait(i18n._("Saving..."), i18n._("Please wait"));
    this.getRpcNode().updateAll(function(result, exception) {
        Ext.MessageBox.hide();
        if(Ung.Util.handleException(exception)) return;
        // exit settings screen
        this.closeWindow();
    }.createDelegate(this), this.getBaseSettings(),
    this.gridFilters.getSaveList());
}
```

21.4.9 补充内容

1. 重要 JavaScript 文件

/src/uvm-lib/servlets/webui/root/script/components.js
- 组件 ConfigItem, AppItem, Node, NodePreview, BlingerManager, SystemStats, ActivityBlinger, SystemBlinger, EditorGrid, SettingsWin, NodeWin, UpdateWindow, ManageListWindow, RowEditorWindow, UsersWindow, GroupsWindow, GridEventLog, JsonListReader, Breadcrumbs 等源代码

/src/uvm-lib/servlets/webui/root/script/md5.js
- MD5 算法的 JavaScript 实现版本

/src/uvm-lib/servlets/webui/root/script/main.js
- 主机框 rack 的代码

/pkgs/untangle-apache2-config/files/var/www/script/i18n.js
- 国际化函数的代码

/pkgs/untangle-apache2-config/files/var/www/script/wizard.js
- 安装向导的代码

2. UVM API 调用

1) Jabsorb 简介

Jabsorb 是轻量级 Ajax/Web 2.0 框架,它允许 Web 浏览器中的 JavaScript 脚本可以调用服务器端 Java Web 应用类的方法。

Jabsorb 采用 JSON-RPC 协议来进行服务器和客户端之间的传输通信。Jabsorb 用来实现 UVM API 的调用。使用 Jabsorb 的 json-rpc 实现版本(http://jabsorb.org/manual)对节点配置,调用 Node API (如为协议控制节点的 ProtoFilter interface 方法)。该接口应该提供相应的方法来显示标页码的数据,更新的表项和事件日志。

目前不为单个 list 单独使用 setter 方法(如 ProtoFilter. updatePatterns(...))。当有多个 list 时,需要逐个更新,再对 Node 调用 reconfigure()函数。相反,我们使用 helper method、updateAll 来更新所有设置(base settings, lists),并且重配置 Node。这种方法在节点配置后只需保存一次。

2) Untangle Jabsorb 使用

(1) jabsorb 的安装。

第 1 步:加入 jar 包。

Jabsorb 需要三个 jar 包:jabsorb-1. 2. 2. jar、slf4j-api-1. 4. 3. jar 和 slf4j-log4j12-1. 4. 3. jar,这三个文件都可以在 Jabsorb-1. 2. 2-src. zip 的压缩包中找到。

将这三个文件放到＜Web 根目录＞WEB-INF 下的 lib 目录中,或＜Tomcat 安装目录＞下的 lib 目录中。

```
/usr/share/untangle/web/webui/WEB-INF/lib
```

第 2 步:配置 web. xml。

web. xml 在以下目录中:

```
/usr/share/untangle/web/webui/WEB-INF/
```

web. xml 文件内容显示如下:

```
<servlet>
   <servlet-name>JSONRPCServlet</servlet-name>
   <servlet-class>org.jabsorb.JSONRPCServlet</servlet-class>
   ⋮
   <init-param>
      <param-name>gzip_threshold</param-name>
      <param-value>0</param-value>
   </init-param>
</servlet>

 ⋮

<servlet-mapping>
   <servlet-name>JSONRPCServlet</servlet-name>
   <url-pattern>/JSON-RPC</url-pattern>
</servlet-mapping>
```

以上配置文件初始化 Jabsorb 引擎 Servlet。其中 gzip_threshold 可以取－1、0 和一个正整数。如果为 0,表示对响应的所有内容进行压缩;如果值为－1,表示不会对响应的内容进行压缩;如果为一个正整数,表示当响应内容超过这个整数时,进行压缩。

当浏览器不支持 gzip 压缩格式,或是经过压缩后的尺寸要比不压缩的尺寸还大时(当响应内容比较少时可能发生这种情况),Jabsorb 就不会对响应内容进行压缩。因此,最好将这个值设为 0,但这样做所付出的代价是可能会对所有的响应内容进行压缩。具体设置可根据具体情况而定。

将 jsonrpc.js 复制到<Web 根目录>script 中，该文件在 Jabsorb 的压缩包中也可以找到。

```
/var/www/jsonrpc/jsonrpc.js
```

(2) 编写 Jabsorb 简单应用程序。

第 1 步　编写一个用客户端访问的 Java 类。

```
package invoke;

public class Message implements java.io.Serializable
{
     public String getMessage(String s)
     {
          return "hello  world"+ s;
     }
}
```

第 2 步　编写 JSP 代码。

```
<%--   index.jsp  --%>

<%@ page language= "java" import= "java.util.* " pageEncoding= "utf- 8"%>

<!DOCTYPE HTML PUBLIC "- //W3C//DTD HTML 4.01 Transitional//EN">
<html>
     <head>
<script type= "text/javascript" src= "script/jsonrpc.js"></script>
<script type= "text/javascript">
function onLoad()
{
     jsonrpc= new JSONRpcClient("JSON- RPC");
}
window.onload= onLoad;

function invoke()
{
     var text= document.getElementById("text");
     var result= jsonrpc.msg.getMessage(text.value);
     alert(result);
}
</script>
<jsp:useBean id= "JSONRPCBridge" scope= "session"
    class= " org.jabsorb.JSONRPCBridge " />
<jsp:useBean id= "message" scope= "session"
     class= "invoke.Message" />
<%
JSONRPCBridge.registerObject("msg", message);
%>
```

```
</head>

    <body>
    <input type= "text" id= "text" />
    <input type= "button" value= "get message"   />
    </body>
</html>
```

在 JSP 文件中需要做如下四件事才能调用 getMessage 方法。

① 引用 jsonrpc.js 文件。

② 在 onLoad 函数中创建 JSONRpcClient 对象。JSONRpcClient 类的构造方法的参数值就是在 web.xml 中配置的 JSON-RPC。

③ 使用<jsp:userBean>创建 org.jabsorb. JSONRPCBridge 和 invoke.Message 对象。

④ 使用 JSONRPCBridge 的 registerObject 方法注册 Message 类,其中第一个参数可以是任意的字符串(这个参数是注册名),第二个参数是 Message 对象实例。registerObject 方法可以对同一个注册名使用多次,但后一个将覆盖前一个对象。

在做完上述工作后,就可以使用 jsonrpc.msg.getMessage 来调用 getMessage 方法。

(3) 在 Servlet 中使用 Jabsorb。

除了在 JSP 中使用 Jabsorb 外,也可以在 Servlet 中使用它。代码如下:

```
public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    HttpSession session= request.getSession();
    JSONRPCBridge bridge= (JSONRPCBridge) session.getAttribute("JSONRPCBridge");
    if(bridge ==null)
    {
        bridge= new JSONRPCBridge();
        session.setAttribute("JSONRPCBridge", bridge);
    }
    bridge.registerObject("msg", message);

}
```

从上面的代码可以看出,在 Servlet 中使用 Jabsorb,实际上就是使用 registerObject 方法来注册 Message 类。然后可以 forward 到使用 Jabsorb 的 JSP 页面,也可以使用 PrintWriter 在当前 Servlet 中输出相应的 JavaScript 和 HTML 代码。

(4) 注册全局对象。

使用 registerObject 注册的对象只能在当前页面中访问。如果想注册一次,就可在以任何运行在当前 Web 服务器的页面(JSP、HTML 等)中使用这个对象,就需要使用如下的代码来注册 Message 对象:

```
JSONRPCBridge.getGlobalBridge().registerObject("globalMsg", message);
```

可以将上面的相应代码换成这行代码,然后另建立一个 test.jsp,然后使用如下的代码

调用 getMessage 方法：

```
<script type="text/javascript" src="script/jsonrpc.js"></script>
<script type="text/javascript">
try
{
    jsonrpc=new JSONRpcClient("JSON-RPC");
    //如果将 globalMsg 换成 msg,将抛出[object error]错误
    var result=jsonrpc.globalMsg.getMessage("bill");
    alert(result);
}
catch(e)
{
  alert(e);
}
</script>
```

Untangle 代码见文件：/src/uvm-lib/servlets/webui/src/com/untangle/uvm/webui/jabsorb/UtJsonRpcServlet.java：

```
public class UtJsonRpcServlet extends JSONRPCServlet
{
    private static final String BRIDGE_ATTRIBUTE="JSONRPCBridge";

     ⋮

    private void initSessionBridge(HttpServletRequest req)
    {
        HttpSession s=req.getSession();
        JSONRPCBridge b=
        (JSONRPCBridge)s.getAttribute(BRIDGE_ATTRIBUTE);


        if (null==b) {
            b=new JSONRPCBridge();
            s.setAttribute(BRIDGE_ATTRIBUTE, b);
            b.setCallbackController(new UtCallbackController(b));

            RemoteUvmContext uvm=
            LocalUvmContextFactory.context().remoteContext();
            b.registerObject("RemoteUvmContext", uvm,
            RemoteUvmContext.class);
```

/src/uvm-lib/servlets/setup/src/com/untangle/uvm/setup/jabsorb/UtJsonRpcServlet.java：

```
protected JSONRPCBridge findBridge(HttpServletRequest request)
{
```

```
        // Find the JSONRPCBridge for this session or create one
        // if it doesn't exist
        HttpSession session= request.getSession( false );
        JSONRPCBridge jsonBridge= null;
        if (session != null) jsonBridge= (JSONRPCBridge)
        session.getAttribute(BRIDGE_ATTRIBUTE);

        if ( jsonBridge == null) {
            /* Use the global bridge if it can't find the session bridge */
            jsonBridge= JSONRPCBridge.getGlobalBridge();
                if ( logger.isDebugEnabled()) logger.debug("Using global bridge.");
            }
            return jsonBridge;
    }


    public class UtJsonRpcServlet extends JSONRPCServlet
    {
        private static final String BRIDGE_ATTRIBUTE= "JSONRPCBridge";

         ⋮

        private void initSessionBridge(HttpServletRequest req)
        {
            HttpSession s= req.getSession();
            JSONRPCBridge b=
            (JSONRPCBridge)s.getAttribute(BRIDGE_ATTRIBUTE);


            if (null == b) {
                b= new JSONRPCBridge();
                s.setAttribute(BRIDGE_ATTRIBUTE, b);
                b.setCallbackController(new UtCallbackController(b));

                RemoteUvmContext uvm=
                LocalUvmContextFactory.context().remoteContext();
                b.registerObject("RemoteUvmContext", uvm,
                RemoteUvmContext.class);
```

/src/mail-casing/servlets/quarantine/src/com/untangle/node/mail/web/euv/UtJsonRpcServlet.java：

```
    public class UtJsonRpcServlet extends JSONRPCServlet
    {
        private static final String BRIDGE_ATTRIBUTE= "JSONRPCBridge";

        // HttpServlet methods ------------------------------------------
```

```
    public void service(HttpServletRequest req, HttpServletResponse resp)
        throws IOException
    {
        initSessionBridge(req);
        super.service(req, resp);
    }

    // private methods ----------------------------------------------

    private void initSessionBridge(HttpServletRequest req)
    {
        HttpSession s= req.getSession();
        JSONRPCBridge b=
        (JSONRPCBridge)s.getAttribute(BRIDGE_ATTRIBUTE);
        if (null ==b) {
            b= new JSONRPCBridge();
            s.setAttribute(BRIDGE_ATTRIBUTE, b);

            b.registerObject("Quarantine",
            JsonInterfaceImpl.getInstance(),JsonInterface.class);
        }
    }
}
```

Untangle 代码见如下文件：

```
/src/uvm-lib/servlets/webui/root/script/main.js:
rpc.jsonrpc= new JSONRpcClient("/webui/JSON-RPC");

/src/uvm-lib/servlets/reports/root/script/reports.js:
rpc.jsonrpc= new JSONRpcClient("/webui/JSON-RPC");

/src/uvm-lib/servlets/setup/root/script/setup.js:
rpc.jsonrpc= new JSONRpcClient( "/webui/JSON-RPC" );

/src/uvm-lib/servlets/setup/root/script/language.js:
rpc.setup= new JSONRpcClient("/setup/JSON-RPC").SetupContext;
```

(5) 访问集合类型。

如果返回的数据很多，可以使用 Java 提供的集合类型，如将 Message 扩展为如下形式：

```
package invoke;

public class Message implements java.io.Serializable
{
    public String getMessage(String s)
    {
```

```
        return "你好   "+ s;
    }
    public java.util.List getList()
    {
        java.util.List list= new java.util.LinkedList();
        list.add("中国");
        list.add(1234);
        return list;
    }

    public java.util.Map getMap()
    {
        java.util.Map map= new java.util.HashMap();
        map.put("bird", "鸟");
        map.put("human", "人类");
        return map;
    }
}
```

index.jsp 中可加入如下代码来访问 getList、getMap 方法中的数据：

```
<script type= "text/javascript">
jsonrpc= new JSONRpcClient("JSON- RPC");

alert(jsonrpc.globalMsg.getList().list[1]);
    alert(jsonrpc.globalMsg.getMap().map['bird']);
</script>
```

（6）异步调用。

以上代码都是同步调用，在返回结果之前，客户端程序会被阻塞。为了在网络环境不畅的环境下 Web 程序仍然能运行良好，这就需要进行异步调用，客户端在发送请求后立即返回。一旦服务端返回信息，才会调用另一个"回调函数"来获取结果。

回调函数必须有两个参数，第一个参数表示返回值，第二个参数表示异常信息。如果无异常信息，第二个参数值为 null。下面是一个回调函数：

```
function asyc(result,e)
{
    if(e ==null)
        alert(result);
}
```

可以使用下面的代码以异步方式来调用 getMessage 方法：

```
jsonrpc.msg.getMessage(asyc, 'bill');
```

从上面的代码可以看出，异步调用和同步调用的区别就是异步调用需要将回调函数作为方法的第一个参数传入被调用的方法。后面跟着被调用方法的参数值。

Untangle 回调函数见：

```
/src/uvm-lib/servlets/webui/src/com/untangle/uvm/webui/jabsorb/
UtCallbackController.java: UtCallbackController(JSONRPCBridge bridge)

/src/uvm-lib/servlets/setup/src/com/untangle/uvm/setup/jabsorb/
UtCallbackController.java: UtCallbackController(JSONRPCBridge bridge)
```

3. Ext JS 资源

Ext JS 开发框架详见 http://www.sencha.com/products/extjs，主要包括 Ext API 的文档和 Document Model 组件模型。

```
Ext API Documentation
Ext Component Model
```

4. JavaScript 与 HTML 调试

Firebug 是由 Joe Hewitt 开发的一组与 Firefox 集成的 Web 开发调试工具。可以通过 Firebug 实时编辑、调试、监测任何页面的 CSS、HTML 和 JavaScript。Firebug lite 版本也可以在 IE 浏览器中使用，功能比较简单。

ExtJS 目录树如下：

```
/var/www/ext/adapter/ext/
/var/www/ext/adapter/jquery/
/var/www/ext/adapter/prototype/
/var/www/ext/adapter/yui/
/var/www/ext/docs/
/var/www/ext/source/
/var/www/ext/build/
/var/www/ext/examples/
/var/www/ext/resources/
ext-all.js   ext-core-debug.js   ext-all-debug.js   ext-core.js
```

21.4.10 I18N

I18N 中文化之前，需要在翻译之前抽取 keys 值。单个设置类在 I18N 属性中自动加载 Ung.NodeI18N 对象 object。对 I18N 内部 node settings 使用 this.i18n._("My English Text")。

详见目录\untangle\src\i18ntools\。

1. 为 po/pot 文件创建目录

需要为每个 node 创建 po 文件目录，目录名称格式如下：

```
<nodePackageName>/po(如 adblocker/po)
```

2. 抽取字符串至 pot 文件

```
xgettext --copyright-holder='Untangle, Inc.' -L Python -ki18n._ -o keys.pot ../hier/usr/share/
```

```
untangle/web/webui/script/untangle-node-adblocker/settings.js
```

使用-j 选项将 messages 加入现有的 message 文件。

```
xgettext -j --copyright-holder='Untangle, Inc.' -L Java -kmarktr -o tmp_keys.pot ../impl/com/
untangle/node/adblocker/*.java
```

3. 从 pot 文件创建 po 文件为合适的语言

创建和合并新翻译文件到 pot 文件：

```
cp keys.pot ro/shield.po
```

如果 po 文件已经存在，使用 msgmerge 命令：

```
msgmerge -U ro/shield.po keys.pot
```

改变 charset 为 UTF-8：

```
"Content-Type: text/plain; charset=UTF-8\n"
```

4. 翻译 keys

为了编辑 po 文件，可采用 gted 工具(GetText Editor)，它是编辑 gettext po 文件的编辑器(如 gted，见 http://www.gted.org/)。该工具非常易用，可以集成在 Eclipse IDE 的插件中。

21.5 Untangle 内核驱动模块编译

参见 wiki Build new Kernel Modules。主要针对网络驱动等应用场景。

(1) 安装 Untangle 系统。

(2) 登录 http://localhost/alpaca，使用默认密码 passwd 配置外部接口。

(3) 删除配置文件/etc/ssh/sshd_not_to_be_run，运行/etc/initd/sshd/start 启用 sshd 服务。

(4) 通过终端登录，ssh 连接 root@ip.addr。

(5) 创建/etc/apt/sources.list.d/ dev.list 文件，内容如下：

```
deb http://ftp.us.debian.org/debian lenny main contrib non-free
```

(6) 运行 apt-get update。

(7) 运行 apt-get --yes install untangle-development-build。

(8) 在 i386 系统，运行 apt-get --yes install linux-headers-2.6.26-1-untangle-486 linux-headers-2.6.26-1-untangle-686。

如果是 amd64 系统，运行 apt-get --yes install linux-headers-2.6.26-1-untangle-amd64。

(9) 立即删除/etc/apt/sources.list.d/dev.list。

这样就可以重新编译一个.ko 内核模块，对于某些驱动程序的编译必不可少。编译后将其复制至/lib/modules 目录下。

21.6 Untangle UI 界面

安全模块主管理界面是基于 Ext JS 库开发,通过 Ext JS 库提供的组件进行扩展堆砌组合而成。

浏览器的 JavaScript 代码基于 Jabsorb 开源库,通过 JSON-RPC 协议调用 this.getRpcNode()方法获得 Untangle 中注册的对应 UVM 模块 Node 的设置和事件信息,可根据每个模块提供的接口进行获取设置和更新设置的操作。

21.6.1 CSS

Untangle UTM 平台界面 UI 主要由 CSS(css 文件)和背景图片文件组成,分别在 css 目录和图片目录下。

```
/pkgs/untangle-apache2-config/files/var/www/skins/default/css/admin.css
/pkgs/untangle-apache2-config/files/var/www/skins/default/css/ext-skin.css
/pkgs/untangle-apache2-config/files/var/www/skins/default/css/reports.css
/pkgs/untangle-apache2-config/files/var/www/skins/default/css/user.css


/pkgs/untangle-apache2-config/files/var/www/skins/default/images
```

其他主题文件分别在以下目录下:

```
/pkgs/untangle-apache2-config/files/var/www/skins/cammo/css
/pkgs/untangle-apache2-config/files/var/www/skins/cammo/images

/pkgs/untangle-apache2-config/files/var/www/skins/defaultwide/css
/pkgs/untangle-apache2-config/files/var/www/skins/defaultwide/images
```

css 文件主要应用于以下主界面文件 main.js 中:

```
/src/uvm-lib/servlets/webui/root/script/main.js
-包含主机框 center(包括 rack-list, rack-nodes), east(包括 appitems, configItems)
的代码
```

21.6.2 main 主页面

当访问 index.jsp 时,Tomcat web 服务器会自动跳转到 startPage.jps,调用 servelt 程序 tartPageServcet.java,生成动态 HTML 网页。

```
\src\uvm-lib\servlets\webui\root\index.jsp
\src\uvm-lib\servlets\webui\root\WEB-INF\jsp\startPage.jsp
\src\uvm-lib\servlets\webui\src\com\untangle\uvm\webui\servlet\
StartPageServlet.java
```

```
\src\uvm-lib\servlets\webui\root\script\main.js
\src\uvm-lib\servlets\webui\root\script\componets.js

\src\uvm-lib\servlets\webui\root\script\config\administration.js
\src\uvm-lib\servlets\webui\root\script\config\localDirectory.js
\src\uvm-lib\servlets\webui\root\script\config\policyManager.js
\src\uvm-lib\servlets\webui\root\script\config\systemInfo.js
\src\uvm-lib\servlets\webui\root\script\config\system.js
\src\uvm-lib\servlets\webui\root\script\config\upgrade.js
```

如在将 CPU 负载显示为百分比，可以修改如下：\src\uvm-lib\servlets\webui\root\script\component.js。

```
toolTipEl.child("span[name=load_average_1_min]").dom.innerHTML=stats.map.oneMinuteLoadAvg;
toolTipEl.child("span[name=load_average_5_min]").dom.innerHTML=stats.map.fiveMinuteLoadAvg;
toolTipEl.child("span[name=load_average_15_min]").dom.innerHTML=stats.map.fifteenMinuteLoadAvg;

this.getEl().child("div[class=cpu]").dom.innerHTML=Math.round(stats.map.oneMinuteLoadAvg*100.0)+"%";
toolTipEl.child("span[name=load_average_1_min]").dom.innerHTML=Math.round(stats.map.oneMinuteLoadAvg*100.0)+"%";
toolTipEl.child("span[name=load_average_5_min]").dom.innerHTML=Math.round(stats.map.fiveMinuteLoadAvg*100.0)+"%";
toolTipEl.child("span[name=load_average_15_min]").dom.innerHTML=Math.round(stats.map.fifteenMinuteLoadAvg*100.0)+"%";
```

如将磁盘空间的大小显示，可以修改如下文件：\src\uvm-lib\impl\com\untangle\uvm\engine\MessageManagerImpl.java。

```
class MessageManagerImpl implements LocalMessageManager
{
    ⋮
    private static final Pattern DISK_STATS_PATTERN
        = Pattern.compile("\\s*\\d+\\s+\\d+\\s+[hs]d[a-zA-Z]+\\d+\\s+(\\d+)\\s+\\d+\\s+(\\d+)");

        ⋮
        private synchronized void getDiskUsage(Map<String, Object> m)
            throws IOException
        {
            File root= new File("/");
            m.put("totalDiskSpace", root.getTotalSpace());
```

```
            m.put("freeDiskSpace", root.getFreeSpace());
        ⋮

}
```

21.6.3 Firebug

Firebug 是由 Joe Hewitt 开发的与 Firefox 集成的 Web 开发工具,可以通过它实时编辑、调试和监测任何的页面的 CSS、HTML 和 JavaScript。

第 22 章　Untangle Linux 内核源码编译

Untangle Linux 内核从 Linux 内核源码编译而来，但是需要 Pactch Netfilter 的两个补丁。

在 Debian Linux 系统上打上 netfilter patch 后，编译内核代码，最终在 boot 目录下生成所需要的启动文件：

config-2.6.26-1-untangle-686，

System.map-2.6.26-1-untangle-686，

vmlinuz-2.6.26-1-untangle-686，

initrd.img-2.6.26-1-untangle-686。

22.1　Untangle Linux 内核生成

从源码编译 Linux 内核（Debian Linux 系统）非生成 deb 包。

编译安装如下。

（1）下载最新的 Linux 内核（http://www.kernel.org）。

现在最新的是 2.6.26，其下载地址是 http://www.kernel.org/pub/linux/kernel。

（2）复制内核文件到/usr/src 目录，并解压。

输入如下命令：

```
cp linux-2.6.26.tar.bz2 /usr/src
cd /usr/src
tar jxvf linux-2.6.26.tar.bz2
```

如果解压报错，安装 bzip2 软件：

```
apt-get install bzip2
```

解压完毕后会生成 linux-2.6.26 目录：

链接到 linux 目录：

```
ln -s /usr/src/linux-2.6.26 /usr/src/linux
```

（3）复制当前配置文件到新内核所在的目录，输入如下命令：

```
cd /usr/src/linux
cp /boot/config-2.6.18-4-686 .config
```

（4）重新编译你的内核，输入如下命令：

#安装 build-essential 包。否则报错：make[1]：*** [scripts/basic/fixdep] Error 1。

```
apt-get install build-essential gcc g++ libncurses5-dev  initrd-tools
```

```
make mrproper
```

该命令确保源代码目录下没有不正确的.o文件以及文件的互相依赖。

```
make menuconfig
```

配置新内核，我这里用的是旧的配置文件。

```
make
```

此命令完成了 make bzImage 和 make modules 的工作。

(5) 安装新内核。

```
make modules_install
```

把内核模块安装到/lib/modules/2.6.x下。

```
make install
```

完成 mkinitrd 命令及内核(bzImage)和 System.map 的复制。如果系统使用的是 Grub，会自动修改引导选项；对于 LILO 则要手工修改，重写引导记录。

如果以上操作没有生成/boot/initrd.img 文件，进行下面两步。

(6) 生成 initrd.img 文件，输入如下命令：

```
mkinitrd -o /boot/initrd.img-2.6.26 /lib/modules/2.6.26
```

此时如果报错：-bash：mkinitrd：command not found。

```
apt-get install initrd-tools
```

此命令会在/boot 目录下生成 initrd.img-2.6.26 文件。

更改根目录 initrd.img 和 vmlinuz 指向。

```
cd /
rm initrd.img vmlinuz
ln -s /boot/initrd.img-2.6.26 initrd.img
ln -s /boot/vmlinuz-2.6.26 vmlinuz
```

(7) 更新 Grub，输入如下命令：

```
update-grub
```

22.2 Untangle Patched Linux 内核源码

Untangle 在 Linux 内核源码基础上做了增强，此时需要安装 Untangle 源码的 netfilter patch 用于网络协议处理。

第 23 章　Untangle 新版本定制

在遵守 Untangle 公司 GPLv2 许可的情况下，可以自己定制生成自有 Untangle 版本。

23.1　Untangle 安装光盘创建

Debian Linux 本身是基于 Linux 内核的发行版，Debian Linux 系统鼓励在 Debian 的基础上定制自己发行版。

定制 Untangle 发行版需要创建 debian-installer（简称 d-i）。d-i 自身依赖于 udeb 包，udeb 包一部分在光盘中，一部分在 initrd 中。udeb 包又称 micro deb 包，是一种简化的 debian 软件包，不包含 md5sum、doc、man，不能作为普通 deb 包进行安装，主要是为 debian-installer 使用。

D-I 主要为 Debian 初始系统使用，一旦完成安装后，Debian Linux 系统就可以采用 APT（Advanced Package Tool）来管理 deb 包了。

将新开发的 ServiceNode 和 FilterNode 的 deb 集成到 Untangle 安装光盘中，方便为更多的用户使用，这就生成了 Untangle Debian Linux 发行版。

23.2　Untangle 安装光盘修改

修改 Untangle 安装光盘的过程如下。

（1）挂载原 ISO，脚本如下：

```
cd /mnt
mkdir -p /mnt/iso
mount -t iso9660 -o loop image.iso /mnt/iso
```

（2）创建光盘内容的副本，用于修改，脚本如下：

```
cd /mnt
rm -rf image
cp -a iso image
```

（3）增加额外 deb 包到光盘源。

将 additional 中所有 deb 安装包以及放入/mnt/image/pool/main/additional 中，包括新编译的内核包 linux-image-2.6.26-1-XXX-686.deb（XXX 为自己标记名，如 THU）。

注意： 由于编译出的 src 软件包版本低于光盘中的版本号，所以应删除/mnt/image/pool/main/u/untangle-vm 中的 untangle-libuvm、untangle-libuvmcore、untangle-libthirdparty 和 untangle-vm 四个包，或者在 src/deb/changelog 中把版本号修改为更高的版本（高于 svn20090924r24591）。

（4）默认内核与软件包的修改。

修改的目标文件为/mnt/image/ simple-cdd 目录中的 default. preseed 和 expert. preseed 文件，做以下修改：

```
base-installer base-installer/kernel/image string linux-untangle
```

修改为

```
base-installer base-installer/kernel/image string linux-image-2.6.26-1-XXX-686
```

以及

```
d-i pkgsel/include string untangle-gateway
```

修改为（即包含所有 additional 中的包）

```
d-i pkgsel/include string untangle-gateway libsys-hostname-long-perl
libnetpbm10 libmicrohttpd4 openvpn-blacklist untangle-buildutil libdigest-sha1-perl
iceweasel-l10n-zh-cn libdjvulibre21 libnet-ident-perl libgraphviz4 lha netpbm unrar
libsocket6-perl libnetaddr-ip-perl libjasper1 liburi-perl libpkcs11-helper1
libmailtools-perl libhtml-tree-perl liberror-perl openvpn re2c giflib-tools libwww-
perl untangle-spamassassin-update libilmbase6 libpcre3 libgd2-xpm djvulibre-desktop
spamassassin arj gocr gcc untangle-fuzzyocr zoo libpcrecpp0 libnet-dns-perl libclamav5
libnet-ip-perl libjson0 razor libmail-spf-perl libhtml-tagset-perl gcc-4.3
libdigest-hmac-perl untangle-webfilter-init transfig libopenexr6 libjpeg-progs gawk
libungif-bin libfont-afm-perl liblzo2-2 libimage-exiftool-perl libwmf0.2-7
libimage-exif-perl libhtml-format-perl imagemagick libmagick10 gsfonts libstring-
approx-perl libhtml-parser-perlXXX-
node-reporting untangle-node-router XXX-base-spam XXX-node-ips XXX-node-clam XXX
-node-shield XXX-node-spamassassin XXX-base-virus XXX-node-firewall XXX-
base-webfilter XXX-node-openvpn XXX-node-collaborator XXX-node-phish XXX-node-
webfilter XXX-node-sniffer XXX-node-adblocker XXX-node-protofilter XXX-node-
spyware XXX-fontuntangle-libuvmcore untangle-casing-mail untangle-vm untangle-
libuvm untangle-casing-http untangle-casing-ftp untangle-libuvmthirdparty untangle-
buildutil untangle-snort-rules untangle-bootsplash untangle-shield untangle-restore
-tools untangle-hardware-config clamav-freshclam untangle-
apache2-config untangle-net-alpaca untangle-net-alpaca-iptables untangle-kiosk
clamav-base untangle-clamav-config clamav clamav-daemon
```

（5）更新光盘源的 Release 和 Packages 文件。

创建文件/mnt/config-deb 内容如下：

```
Dir {
     ArchiveDir "image";
     OverrideDir "indices";
     CacheDir "indices";
};
TreeDefault {
     Directory "pool/";
```

```
};
BinDirectory "pool/main" {
    Packages "dists/lenny/main/binary- i386/Packages";
    BinOverride "override";
    ExtraOverride "override.extra";
};
Default {
    Packages {
        Extensions ".deb";
    };
};
```

创建文件 config-udeb，内容如下：

```
Dir {
    ArchiveDir "image";
    OverrideDir "indices";
    CacheDir "indices";
};
TreeDefault {
    Directory "pool/";
};
BinDirectory "pool/main" {
    Packages "dists/lenny/main/debian- installer/binary- i386/Packages";
    BinOverride "override";
    ExtraOverride "override.extra";
};
Default {
    Packages {
        Extensions ".udeb";
    };
};
```

创建文件 config-rel，内容如下：

```
APT::FTPArchive::Release::Codename "lenny";
APT::FTPArchive::Release::Origin "Debian";
APT::FTPArchive::Release::Components "main";
APT::FTPArchive::Release::Label "Debian";
APT::FTPArchive::Release::Architectures "i386";
APT::FTPArchive::Release::Suite "Stable";
```

创建文件/mnt/indices/override 和 /mnt/indices/override. extra，为空即可，脚本如下：

```
touch/mnt/indices/override
touch/mnt/indices/override.extra
```

更新 Release 和 Package，脚本如下：

```
apt- ftparchive generate config- udeb
apt- ftparchive generate config- deb
apt- ftparchive - c config- rel release image/dists/lenny/> | image/dists/
lenny/Release
```

(6) 更新 md5 值，脚本如下：

```
cd image
rm md5sum.txt
md5sum 'find ! - name "md5sum.txt" ! - path "./isolinux/* " - follow - type f' > | md5sum.
txt
```

(7) 创建新的 iso 文件，脚本如下：

```
mkisofs - o new.iso - r - b isolinux/isolinux.bin - c isolinux/boot.cat - no- emul- boot
- boot- load- size 4 - boot- info- table /mnt/image
```

注意：可能需要事先安装 mkisofs 的软件包，即 apt-get install mkisofs 具体可参考 debian wiki 的内容。

23.3 Untangle 安装启动背景定制

为了增加界面友好性，可以定制特有的风格背景。

1. Boot Screen 修改

安装时出现选择框和背景图片，修改。

```
/isolinux/isolinux.cfg

default vesamenu.c32
prompt 0
timeout 0

MENU TITLE THU- NSLAB installer
MENU BACKGROUND /images/Untangle.png

label gui
        menu label ^Graphical install (normal mode)
        kernel /install.386/vmlinuz
        append  preseed/file= /cdrom/simple- cdd/default.preseed
        video= vesa: ywrap,mtrr vga= 788 initrd= /install.386/gtk/initrd.gz - - quiet

label gui- expert
        menu label ^Graphical install (expert mode)
        kernel /install.386/vmlinuz
        append  preseed/file= /cdrom/simple- cdd/expert.preseed
```

```
    video= vesa:ywrap, mtrr vga= 788 initrd= /install.386/gtk/initrd.gz - - quiet

label text
    menu label ^Text install (normal mode)
    kernel /install.386/vmlinuz
    append  preseed/file= /cdrom/simple- cdd/default.preseed
    video= vesa: ywrap,mtrr vga= 788 initrd= /install.386/initrd.gz - - quiet
    label text- expert
    menu label ^Text install (expert mode)
    kernel /install.386/vmlinuz
    append  preseed/file= /cdrom/simple- cdd/expert.preseed
    video= vesa:ywrap,mtrr vga= 788 initrd= /install.386/initrd.gz - - quiet
```

修改光盘标志图片，替换为

```
/cdrom.ico
```

2. Grub 选单背景

将 Untangle 中 splashimage=(hd0,0)/boot/grub/utsplash. xpm. gz(/boot/grub/menu. lst) 中这一行注释或删除，或者用新图片制作一个 xpm 文件并压缩，替换以下文件/boot/grub/utsplash. xpm. gz。

3. Untangle 启动进度条前闪现的背景

将/boot/initrd. img-2. 6. 26-1-untangle-686 解压并替换其目录 usr/share/splash/theme/untangle/ 下的三个 png 片，再重新压制为 initrd. img，操作方法见 initrd/newinitrd. sh，而 initrd. img-2. 6. 26-1-XXX-686 是一个已修改好的 img 文件。

23. 4 Untangle 安全组件 ICON 图标

重新设计各种安全组件的 ICON 图标，形成自有风味。

(1) 选择一张 PNG 格式的图片，Untangle 里用的是 42×42 的，如文件 1. png。

(2) 在 Linux 系统中运行 cat 1. png|base64>1. tmp，把 1. png 的字节流转成 ASCII 表示的形式，存在 1. tmp 中。

(3) 把 1. tmp 的文本替换 src/debian/control 中对应 debian 包的 XB-Desc-Icon 属性。

(4) 在 Untangle 系统中重新编译 Debian 包并安装(export SRC_HOME=.;debuild -us -uc -d)。

附录 A　Untangle Upstream Projects

见 http://wiki. untangle. com/index. php/Upstream_Projects。

表 A-1　项目名称及信息

项目名称	许可证	是否可链接	是否可修改	用　途	网　址
Linux Kernel	GPL		Yes	Server	http://kernel. org
Debian Sarge	GPL/ LGPL			Server	http://debian. org
Knoppix	GPL			Server	http://www. knoppix. net/
Iptables	GPL		Yes	Server	http://netfilter. org
Libsysfs	LGPL	Yes		UVM	http://linux-diag. sourceforge. net/ Sysfsutils. html
Hibernate	LGPL			UVM	http://hibernate. org
Postgres	BSD			UVM	http://postgresql. org
Itext	LGPL or MPL	Yes		Reports, GUI	http://www. lowagie. com/iText/
Jakarta Log4j	Apache	Yes		UVM	http://logging. apache. org/log4j/ docs/
Jakarta Tomcat	Apache	Yes		UVM	http://jakarta. apache. org/tomcat/index. html
Java Activation Framework	Apache	Yes		UVM	http://java. sun. com/products/ javabeans/glasgow/jaf. html
Jfreechart	LGPL	Yes		Reports, GUI	http://www. jfree. org/jfreechart/
Javamail	Sun	Yes		UVM	http://java. sun. com/products/ javamail/
Libxml2	MIT	Yes		UVM	http://xmlsoft. org/
Javawebstart	Sun	Yes		UVM	http://java. sun. com/products/ javawebstart/
Jasper Reports	LGPL	Yes		Reports	http://jasperreports. sourceforge. net/
Kunststoff	LGPL	Yes		GUI	http://www. incors. org/archive/
Netbeans	Sun Public	Yes		GUI	http://www. netbeans. org/
PostgresjDBC	BSD	Yes		UVM	http://jdbc. postgresql. org/
Gnu trove	Lgpl	Yes		UVM	http://trove4j. sourceforge. net/

项目名称	许可证	是否可链接	是否可修改	用　途	网　址
Apache ant	apache	Yes		Build	http://ant. apache. org/
Java JRE 1. 5	Sun			UVM/GUI	http://sun. com/java
I7-Filter	GPL			Protocol Control	http://l7-filter. sourceforge. net/
Clamav	GPL			Virus Blocker	http://www. clamav. net/
Spamassassin	Apache			Spam Blocker	http://spamassassin. apache. org/
Spyware Cookie List	Feel Free to Spread			Spyware	http://www. geocities. com/spywarekilla/autoinstaller. zip
Spyware IP List	Public Domain			Spyware	http://www. geocities. com/yosponge/blockips. txt
Velocity	Apache			Mail-casing	http://jakarta. apache. org/velocity/
Xdoclet	Custom			Build	http://xdoclet. sourceforge. net/xdoclet/index. html
AJAXTK	Mozilla Public License			Not Used Yet	http://www. zimbra. com/communi-ty/ajaxtk_download. html
Bcel	Mozilla Public License			Not Used Yet	http://jakarta. apache. org/bcel/man-ual. html
C3p0	LGPL	Yes		UVM	http://sourceforge. net/projects/c3p0
Jcifs	LGPL	Yes		Remote Access Portal	http://jcifs. samba. org/
OpenVPN	GPL			Openvpn	http://openvpn. net/
Berkeley DB	Oracle/Sleepycat	Yes		Web Filter	http://www. oracle. com/database/berkeley-db/index. html
kaspersky	Kaspersky OEM (Pay)			Kaspersky Virus Blocker	http://kaspersky. com/
Urlblacklist. com	Urlblacklist. com(pay)			Web Filter	http://urlblacklist. com/
Spring MVC	Apache			Not Used Yet	http://www. springframework. net
Ext JS	LGPL			Not Used Yet	http://extjs. com
SARE Rules	Artistic/Gpl Dual			Spam Blocker	http://www. rulesemporium. com/
SANE Security Sigs	Public Domain			Phish Blocker	http://www. sanesecurity. com/clamav/index. htm
Ruby	Ruby License			Network Configuration	http://www. ruby-lang. org

项目名称	许可证	是否可链接	是否可修改	用　途	网　址
Ruby on Rails	MIT			Network Configuration	http://www.rubyonrails.org/
Mongrel	Ruby License			Network Configuration	http://www.rubyonrails.org/
libnfnetlink	GPL	Yes		UVM	http://www.netfilter.org/projects/libnfnetlink/index.html
libnetfilter_queue	GPL	Yes		UVM	http://www.netfilter.org/projects/libnetfilter_queue/index.html
libnetfilter-conntrack	GPL	Yes	Yes	UVM	http://www.netfilter.org/projects/libnetfilter_conntrack/index.html
sqlite	Public Domain	No	No	UVM	http://www.sqlite.org/

表 A-1 中出现的个别词组含义如下。

GPL：General Public License，GNU 通用公共授权。

LGPL：GNU Lesser General Public License，GNU 宽通用公共许可证。

BSD：Berkeley Software Distribution，伯克利软件套件。

MPL：The Mozilla Public License，Netscape 的 Mozilla 小组为其开源软件项目设计的软件许可证。

Apache：非盈利开源组织 Apache 采用的开源协议。

Sun：Sun 公司是 IT 及互联网技术服务公司(已被甲骨文公司收购)。

MIT：源自麻省理工学院(Massachusetts Institute of Technology，MIT)，又称“X 条款”(X License)或“X11 条款”(X11 License)。

Feel Free to Spread：随时蔓延。

Public Domain：公有领域。对于领域内的知识财产，任何个人或团体都不具所有权益。

Mozilla Public License：Mozilla 公共许可证，即 MPL。

Oracle/SleepyCat：Oracle 公司与其收购的开源嵌入式数据库公司 Sleepycat 兼容许可。

Kaspersky OEM (pay)：卡巴斯基实验室许可证。

Urlblacklist.com (pay)：URL 黑名单版权许可。

Artistic /GPL dual：原始艺术许可证与 GPL 兼容许可。

Ruby License：红宝石版权许可。

Server：服务器。

UVM：Universal Verification Methodology，通用验证方法学。

Reports：报告。

GUI：Graphical User Interface，图形用户界面。

Build：Apache Ant 是一个将软件编译、测试、部署等步骤联系在一起加以自动化的一个工具，默认情况下，它的 buildfile(XML 文件)名为 build.xml。

Protocol Control：协议控制。
Virus Blocker：病毒防护。
Spam Blocker：垃圾邮件防护。
Spyware：间谍软件。
Mail-casing：邮件保护壳。
Not Used Yet：未被使用。
Remote Access Portal：远程访问门户。
Openvpn：用于创建虚拟专用网络加密通道。
Kaspersky Virus Blocker：卡巴斯基病毒防护。
Web Filter：网站过滤。
Phish Blocker：钓鱼防护。
Network Configuration：网络配置。
HTTPS Inspector：安全 HTTPS 解剖器。

附录 B　Untangle Java Open Source Collections

1. Apache Tomcat

Apache Tomcat 是一个开源的 Java Servlet 和 JavaServer Pages 软件，在 Java Community Process 这个组织的领导下实施开发工作。

Apache Tomcat 的网址见 http://tomcat.apache.org/。

Tomcat 的主要文件如下：

```
\src\uvm-lib\lib\tomcat-ajp.jar
\src\uvm-lib\lib\tomcat-coyote.jar
\src\uvm-lib\lib\tomcat-http.jar
\src\uvm-lib\lib\tomcat-util.jar
```

2. Hibernate

Hibernate 是一个强大的、针对高性能对象/关系持久性和查询的服务。Hibernate 允许人们依据面向对象的惯用语法开发持久化类，包括关联、继承、多态、组合、集合。Hibernate 允许人们在其自身的便携式 SQL 扩展(HQL)来进行查询，同时也支持本地的 SQL，或一个具有面向对象的标准和 API 的实例。

Hibernate 的网址见 https://www.hibernate.org/。

Hibernate 的主要文件如下：

```
\src\uvm-lib\lib\hibernate-3.2.4.sp1.tar.gz
\src\uvm-lib\lib\hibernate-annotations-3.3.0.GA.tar.gz
\src\uvm-lib\lib\hibernate-client.jar
```

3. Apache commons

Apache commons 是一个 Apache 项目，其关注各方面可重用的 Java 组件。

Apache commons 的网址见 http://commons.apache.org/。

Commons 的主要文件如下：

```
\src\uvm-lib\lib\commons-beanutils-1.7.0.tar.gz
\src\uvm-lib\lib\commons-codec-1.3.tar.gz
\src\uvm-lib\lib\commons-digester-1.7.tar.gz
\src\uvm-lib\lib\commons-fileupload-1.1.tar.gz
\src\uvm-lib\lib\commons-httpclient-3.0.tar.gz
\src\uvm-lib\lib\commons-io-1.1.tar.gz
```

4. Apache 日志记录服务

Apache 日志记录服务项目创建和维护与开源软件相关的应用程序记录，并免费向公众发布。Apache 日志记录服务项目的产品包括三个日志框架：与 Java 相关的 log4j，与 C++ 相关的 Log4cxx，与微软.NET 框架相关的 log4net 以及日志查看和分析工具 Chainsaw。

Apache 日志记录网址见 http://logging.apache.org/。
Apache 日志记录服务的主要文件如下：

```
\src\uvm-lib\lib\logging-log4j-1.2.14.tar.gz
```

5. Apache Byte Code Engineering Library

Byte Code Engineering Library 是为了方便用户去分析、建立和操纵(二进制化)Java 的类文件(即那些以.class 为后缀名的文件)。

其网址见 http://jakarta.apache.org/bcel/index.html。

其主要文件如下：

```
\src\uvm-lib\lib\bcel-M.N.P.tar.gz
```

6. JFreeChart

JFreeChart 是一个免费的(LGPL)Java(tm)平台图表库。

其网址见 http://www.jfree.org/。

其主要文件如下：

```
\src\uvm-lib\lib\jfreechart-M.N.P.tar.gz
\src\downloads\jfreechart-M.N.P.jar
```

7. c3p0

c3p0 是一个易于使用的库，其通过 JDBC 3 规范定义的功能和 JDBC 2 可选的扩展来增强自身，以使传统的 JDBC 驱动能够达到“企业级”。

其网址见 http://www.mchange.com/projects/c3p0/index.html#contents。

其主要文件如下：

```
\src\uvm-lib\lib\c3p0-0.9.0.4.jar
\src\downloads\c3p0-0.9.0.4.bin.tar.gz
```

8. iText

iText 是一个库，它允许人们实时地生成 PDF 文件。

其网址见 http://itextpdf.com/。

其主要文件如下：

```
\src\downloads\itext-1.3.jar
\src\uvm-lib\lib\itext-1.3.jar
```

9. Jabsorb

Java 到 JavaScript 对象请求代理。

其网址见 http://jabsorb.org/。

其主要文件如下：

```
\src\uvm-lib\lib\jabsorb-1.2.2.jar
\src\downloads\jabsorb-1.2.2-src.zip
```

10. JAF

随着 JavaBeans Activation Framework(JAF)标准的扩展,使用Java技术的开发人员可以利用标准服务确定一个任意类型的数据块,封装访问,发现可用的操作,并以适当的对象实例执行操作。例如,如果浏览器取得一个 JPEG 图像,则此框架将令浏览器对该数据流识别为 JPEG 图像,从该类型出发,浏览器可以定位和实例化一个可以操纵或可查看该图像的对象。

其网址见 http://java.sun.com/javase/technologies/desktop/javabeans/jaf/downloads/index.html#download。

其主要文件如下:

```
\src\uvm-lib\lib\actication.jar
\src\downloads\jaf-1.0.2.tar.bz2
```

11. JavaMail API

JavaMail API 提供了一个独立平台和独立协议的框架来建立邮件和消息的应用。

其网址见 http://kenai.com/projects/javamail 和 http://java.sun.com/products/javamail/。

其主要文件如下:

```
\src\downloads\javamail-1_3_3_01.zip*
\src\uvm-lib\lib\mail.jar
\src\uvm-lib\lib\mailapi.jar
```

12. Berkeley DB Java Edition

其网址见 http://www.oracle.com/technology/documentation/berkeley-db/je/installation.html#installJE。

其主要文件如下:

```
\src\downloads\je-3.2.74.tar.gz
\src\uvm-lib\lib\je-3.2.74.jar
```

13. PostgreSQL JDBC driver

PostgreSQL JDBC driver 允许 Java 程序使用标准的、独立于数据库的 Java 代码连接到 PostgreSQL 数据库。它是一个纯 Java(Ⅳ型)的执行。

其网址见 http://jdbc.postgresql.org/index.html。

其主要文件如下:

```
\src\downloads\postgres-jdbc-7.4_215.tar.bz2
\src\uvm-lib\lib\ pg74.215.jdbc3.jar
```

14. JCIFS

JCIFS 是一个 100%使用 Java 开发的开源客户端库,其实现了 CIFS/ SMB 网络协议。CIFS 是微软公司 Windows 平台上的标准文件共享协议。

其网址见 http://jcifs.samba.org/。

其主要文件如下：

```
\src\downloads\jcifs-1.2.9.jar
\src\uvm-lib\lib\jcifs_1.2.9.tar.gz
```

15．properJavaRDP

properJavaRDP 是为 Windows 终端服务开发的一个开源 Java RDP 客户端。它基于 SourceForge 上一个名为 rdesktop 的项目。

其网址见 http://properjavardp.sourceforge.net/。

其主要文件如下：

```
\src\uvm-lib\lib\properJavaRDP-1.1.jar
```

16．Kunststoff

Kunststoff 是为 Java Swing 应用程序开发的一个完全免费的 Look & Feel（软件开发术语：观感）。它支持许多微件（如 JInnerFrame、JProgressBar 等）。

其主要文件如下：

```
\src\uvm-lib\lib\kunststoff-2_0_1.tar.bz2
```

17．AjaxTK JavaScript development library

其主要文件如下：

```
\src\uvm-lib\lib\AjaxTK-3.1.1_GA_394-src.tar.gz
```

18．Alloy Look and Feel

Alloy Look and Feel 是一个专业的 Java Swing 应用程序观感。

其主要文件如下：

```
\src\uvm-lib\lib\alloylnf-1_4_4-1.zip
```

19．Simple Logging Facade for Java（SLF4J）

The Simple Logging Facade for Java or（SLF4J）以一个简单的外观或抽象化服务于各种日志框架，如 java.util.logging、log4j 和 logback，并允许一个使用该产品的用户在部署时可以插入所需的日志框架。

其网址见：http://www.slf4j.org/。

其主要文件如下：

```
\src\uvm-lib\lib\slf4j-1.4.3.tar.gz
```

20．JSON-RPC for Python

JSON-RPC 是轻型远程过程调用协议，类似于 XML-RPC，由 Python 语言开发。

其网址见：http://json-rpc.org/。

其主要文件如下：

```
\src\uvm-lib\lib\python-jsonrpc-r19.tar.gz
```

21．Trove

Trove library 提供 java.util 中 Collections API 的快速实现。

其网址见：http://trove4j.sourceforge.net/。

其主要文件如下：

```
\src\uvm-lib\lib\trove-1.0.2.tar.gz
```

22. Velocity

Velocity 允许人们使用一个简单而强大的模板语言来引用 Java 代码中定义的对象。

其网址见：http://velocity.apache.org/engine/index.html。

其主要文件如下：

```
\src\downloads\velocity-1.4.tar.gz
\src\uvm-lib\lib\velocity-1.4.jar
```

23. XDoclet

XDoclet 使 Java 具有面向属性变成特性。通过向 Java 源中添加元数据(属性)，人们可以使他们的代码更加具有意义。

其网址见：http://xdoclet.sourceforge.net/xdoclet/index.html。

其主要文件如下：

```
\src\downloads\xdoclet-lib-1.2.3.tar.gz
\src\uvm-lib\lib\
```

24. XML-RPC

XML-RPC 是一种远程调用协议，它使用 XML 来编码其呼叫并使用 HTTP 协议作为传输机制。

其网址见：http://www.xmlrpc.com/。

其主要文件如下：

```
\src\downloads\xmlrpc-current-bin.tar.gz
\src\uvm-lib\lib\ xmlrpc-common-3.1.jar
\src\uvm-lib\lib\ xmlrpc-client-3.1.jar
```

25. XStream

XStream 是一个简单的库，它将对象序列化到 XML 后再返回。

其网址见：http://xstream.codehaus.org/。

其主要文件如下：

```
\src\uvm-lib\lib\xstream-distribution-1.3-bin.zip
```

26. YUI Compressor

YUI Compressor 是 JavaScript minifier 为了 100%安全而设计的，它拥有比其他大部分工具更高的压缩率。

其网址见：http://developer.yahoo.com/yui/compressor/。

其主要文件如下：

```
\src\uvm-lib\lib\yuicompressor-2.4.2.zip
```

27．JNPL

JNLP(Java 网络加载协议)是一种基于 XML 的协议，它可以用来在互联网上部署 Java 和 JavaFX 应用程序。

其网址见：http://java.sun.com/developer/technicalArticles/Programming/jnlp/。

其主要文件如下：

```
\src\uvm-lib\lib\jnlp.jar
```

参考文献

[1] Daniel P. Bovet, Marco Cesati. Understanding the Linux Kernel[M]. 3rd Edition Sebastopol. California: O'Reilly Media, 2005.

[2] Christian Benvenuti. Understanding the Linux network Internels[M]. Sebastopol, California: O'Reilly Media, 2009.

[3] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. Linux Device Drivers[M]. 3rd Edition. Sebastopol, California: O'Reilly Media, 2005.

[4] 库察那. Java 软件体系结构设计模式标准指南[M]. 北京：电子工业出版社,2006.

[5] 林胜利. Java 优化编程[M]. 北京：电子工业出版社,2007.

[6] 戈茨. Java 并发编程实践[M]. 北京：电子工业出版社,2007.

[7] Paul Hyde 著. Java 线程编程[M]. 周良忠译. 北京：人民邮电出版社,2003.

[8] 赫姆瑞贾尼. Java 敏捷开发[M]. 北京：人民邮电出版社,2007.

[9] Russ Olsen. Design Patterns in Ruby[M]. Boston: Addison-Wesley Professional,2007.

[10] Jeremy McAnally, Assaf Arkin. Ruby in Practice[M]. Greenwich, Connecticut: Manning Publications, 2009.

[11] Kevin C. Baird. Ruby by Example Concepts and Code[M]. San Francisco: No Starch Press, 2007.

[12] Bruce Tate, Curt Hibbs. Ruby on Rails: Up and Running[M]. Sebastopol, California: O'Reilly Media, 2006.

[13] Obie Fernandez. The Rails Way[M]. Boston: Addison-Wesley Professional, 2007.

[14] A P Rajshekhar. Building Dynamic Web 2.0 Websites with Ruby on Rails[M]. Birmingham: Packet Publishing, 2008.

[15] Mark Lutz. Programming Python[M]. Sebastopol, California: O'Reilly Media,2011.

[16] Alex Martelli, Anna Ravenscroft, David Ascher. Python Cookbook[M]. Sebastopol, California: O'Reilly Media,2005.

[17] Fredrik Lundh. Python Standard Library[M]. Sebastopol, California: O'Reilly Media, 2001.

[18] Alex Martelli. Python In A Nutshell[M]. Sebastopol, California: O'Reilly Media, 2003.

[19] Douglas Crockford. JavaScript: The Good Parts[M]. Sebastopol, California: O'Reilly Media, 2008.

[20] John Resig. Pro JavaScript Techniques[M]. New York City: Apress, 2006.

[21] Danny Goodman. JavaScript & DHTML Cookbook[M]. Sebastopol, California: O'Reilly Media, 2003.

[22] Jason Brittain, Ian F. Darwin. Tomcat: The Definitive Guide[M]. Sebastopol, California: O'Reilly Media, 2003.

[23] Vivek Chopra, Sing Li, Jeff Genender. Professional Apache Tomcat 6[M]. New Jersey: Wrox, 2007.

[24] Hans Bergsten. JavaServer Pages[M]. 3rd Edition. Sebastopol, California: O'Reilly Media, 2003.

[25] Bruce W. Perry. Java Servlet & JSP Cookbook[M]. Sebastopol, California: O'Reilly Media, 2004.

[26] Rich Bowen, Ken Coar. Apache Cookbook[M]. Sebastopol, California: O'Reilly Media, 2008.

[27] Marty Hall. Core Servlets and JavaServer Pages[M]. Upper Saddle River, New Jersey: Prentice Hall PTR, 2000.

[28] James Elliott. Hibernate：A Developer's Notebook [M]. Sebastopol，California：O'Reilly Media，2004.

[29] Jeffrey E. F. Friedl. Mastering Regular Expressions [M]. Sebastopol，California：O'Reilly Media，2006.

[30] Chuck Cavaness. Programming Jakarta Struts[M]. 2nd Edition. Sebastopol，California：O'Reilly Media，2004.